Heterogeneous Graph Based Parallel and Distributed Data Management System for Application in Fraud Detection

Hongming Zhu

A thesis submitted in partial fulfillment of the
requirements of the
University of Bolton
for the degree of
Doctor of Philosophy

This research work was carried out in collaboration with
Tongji University
School of Software Engineering
Shanghai, China

University Reference:	
HESA Reference:	
Program of Study:	PhD
Route of Study:	Research
Student ID:	1009677

Nov 2017

Abstract

With the rapid development of IT technologies, most organizations have their own IT system to record different kinds of data in an organization. Individuals also collect and share data through the Internet. How to manage such a large amount of data and how to make a good use of it becomes a key challenge in knowledge management. As datasets become larger and more complex the ability to process them efficiently without loss or misinterpretation of information becomes a major challenge with current knowledge management systems. The thesis describes a knowledge management system, which attempts to overcome these problems by developing a new and innovative system architecture and algorithms which will enable distributed data storage and parallel processing. The contribution to knowledge will be: 1). develop a basis for data management system architecture based on heterogeneous graph, distributed storage and parallel processing technologies. 2). Develop a basis for knowledge finding an sharing system based on entropy based clustering and attribute selection technologies. 3). Develop a fraud detection prototype system which will used the proposed data management and knowledge finding method.

The developed new and innovative system architecture consists of two parts: storage subsystem and knowledge clustering subsystem. The storage system focuses on two key issues: avoidance of data duplication, and optimization for parallel processing. Since the volume of the dataset may be very large, the storage sub-system has to avoid data duplication and needs to be located quickly. This is achieved by the use of multi dataset schemas to describe the dataset and index the dataset. The index associated with each schema enables the data to be located rapidly. In order to optimize for parallel processing, distributed storage and index technologies are incorporated into the system.

In the knowledge clustering subsystem, a heterogeneous graph is used to describe a body of knowledge with the node representing individual components of the knowledge and the edge representing the linkage between those components. The thesis proposes a feature selection model, combines the graph attribute and graph structure together. The model for the heterogeneous knowledge graph can deal with the incomplete attributes across knowledge and different types of link, according to a user

specified attribute parameters.

The thesis gives an example of a prototype knowledge management system for fraud detection to combine all the ideas together and evaluate all the proposed ideas.

Acknowledgements

This research was completed in University of Bolton (UK) for the degree of Doctor of Philosophy from September 2010 until May 2017. It was also supported by School of software engineering, Tongji University, China.

I would like to thank my supervisors, Prof. Dr. Danny Morton, Stan Oliver from the University of Bolton. Thanks for their invaluable support and patience in the entire research period. Thanks for letting me work and learn under their great supervision. I really appreciate their generosity in providing great cooperation and invaluable suggestion.

My appreciation also goes to Prof. Dr. Gill Green, Paul Oliver, Ling Wu, and my colleagues in School of Software Engineering, Tongji University. I would like to particularly thank Prof. Qin Liu and Mrs. Xiaowen Yang for their time and support, especially in difficulty situations. I would also like to thank Mr. Wenjun Zhou and Mr. You Zhou for their helpful cooperation, technical support and constructive discussions.

Finally, I would like to express my thanks to My wife Guanyu Li and my family for their help.

Declaration

"No portion of the work presented in this dissertation has been submitted in support or another award or qualification either at this institution or elsewhere"

(Hongming Zhu)

Table of Contents

List of Figures

List of Tables

List of Equations

1. Introduction

The technical background and principle concepts of data management are discussed in this introduction. This chapter explains the problem status and research aims for data management. Moreover, there are some critical issues, which should be taken into consideration. These problems, as well as the motivation of the research, will be stated in this chapter. Additionally, the research aims and contributions are illustrated. Finally, the dissertation structure is shown.

1.1. Data Management in a big data environment

Over the last few years Computational Science has been evolving to include information management. Scientists are faced with mountains of data that stem from four trends: (1) the flood of data from new scientific instruments driven by Moore's Law – doubling their data output every year or so; (2) the flood of data from simulations; (3) the ability to economically store petabytes of data online; and (4) the Internet and computational Grid that makes all these archives accessible to anyone anywhere exacerbating the replication, creation, and recreation of more data. (Bell, Gray et al. 2006)

In the business world, "Open Innovation: The New Imperative for Creating and Profiting from Technology" was first published by Henry Chesbrough(Chesbrough 2003). It coined the term "Open Innovation" and explained its application to managerial problems. The term open innovation is based on the assumption that good ideas are just as likely to come from outside an organisation as they are from within. A number of publications have addressed the issue that organisations should be open to generating ideas from outside of the organisation. e.g. Rigby(Rigby and Zook 2002), and Christensen(Christensen, Olesen et al. 2005). Based on the open innovation ideas, data management becomes a key area in open innovation(Fienberg and Martin 1985, Stefansson 2002, Venugopal, Buyya et al. 2006). A lot of research in Content Delivery Network, Peer-to-Peer Network, Distributed File System and Distributed Database has been done in recent years.

Problems of data management can be divided into three different layers: Data

Storage, Data Modeling and Data Analysis.

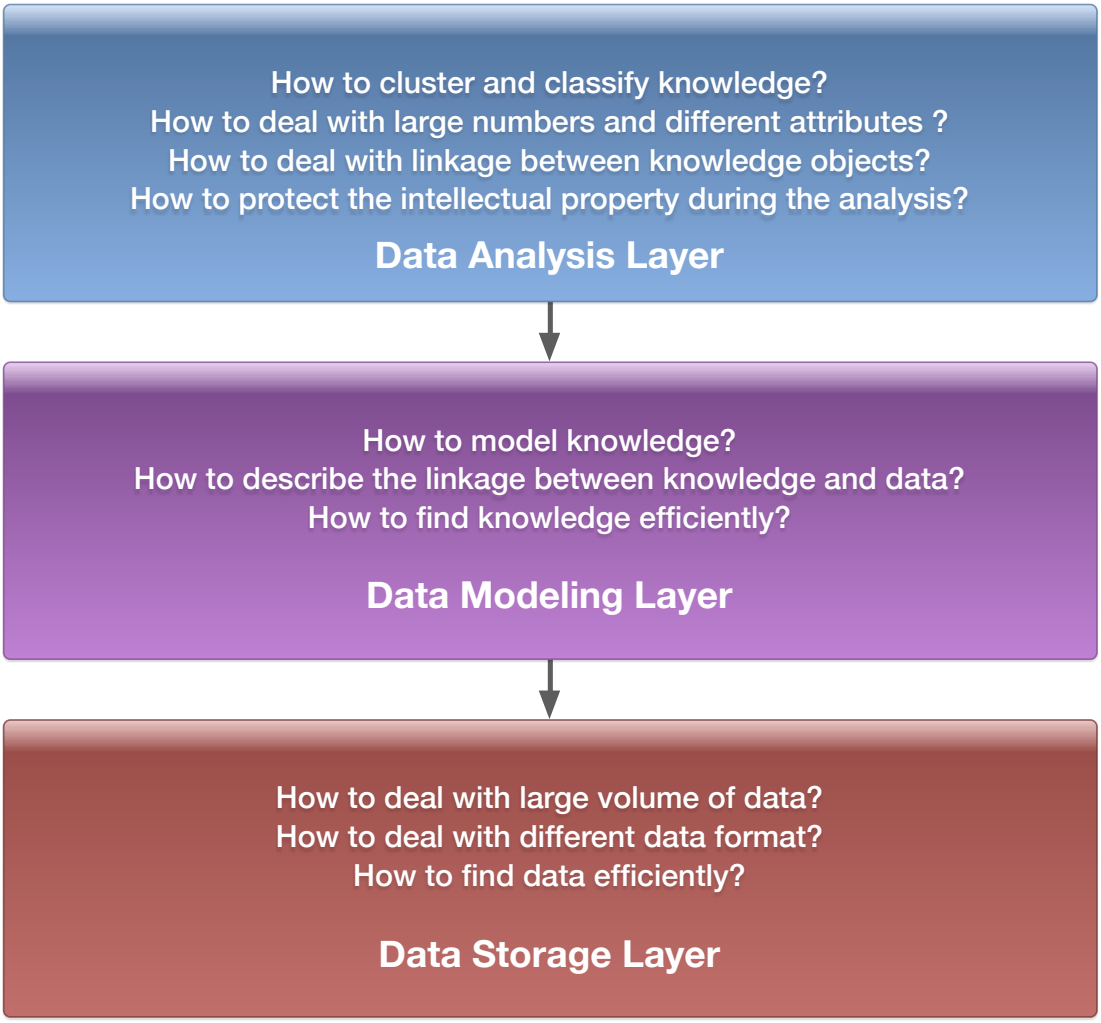


Fig 1. Research Problems in data management and knowledge sharing

The data storage part will focus on how to deal with the large volume data sets from different organizations and individuals in different formats, how to store the dataset, the proper meaning of the dataset, and how to find what you want in the dataset quickly. In the real world, the dataset may come from different sources with different formats, and varying degrees of structure. Some of the datasets are well structured, but there is also a lot of unstructured data arising from sources such as wiki data, web data and social network data. In different applications, multiple sematic meanings or knowledge will be added to the same dataset; different applications may interpret or use the dataset in different ways. Acquisition, organization, query, and visualization tasks needs to be done effectively with large data volumes. All these jobs need to be solved within fixed times (minutes or hours).

The data modeling part will focus on how to organize the data and process it. Most

of the data or informational objects, individual agents, groups, or components are interconnected or interact with each other, forming numerous, large, interconnected, and sophisticated networks. There are different models to present the network of data objects and the connection. The Heterogeneous graph model is one of the choices. In the heterogeneous graph, the objects (i.e., nodes) are of different types, and links among objects corresponding to different relations, denoting different interaction semantics. An object is usually associated with some attributes. For example, in the case of the YouTube social media network, the object types include videos, users, and comments; links between objects correspond to different relations, such as publish and like relations between users and videos, post relation between users and comments, friendship and subscribe relations between users, and etc.; and attributes include user's location, video's clip length and number of views, comments, etc.

The data analysis part will focus on how to analyze, classify, and cluster the data. Whilst there has been a lot of research on data clustering and data classification, in heterogeneous graph based data clustering and data classification, there are still some special challenges. First, an object may contain only partial or even no observations for a given attribute set that is critical to determine their cluster labels. That is, a pure attribute-based clustering algorithm cannot correctly detect these clusters. Second, although links have been frequently used in networks to detect clusters (Clauset, Newman et al. 2004, Von Luxburg 2007) in recent research, a much more challenging scenario in which the links are of different types and interpretations are considered, each of which may have its own level of semantic importance in the clustering process. Consequently, a pure link-based clustering without any guidance from attribute specification could fail to meet user demands.

1.2. Problem statement

Many cloud-based data management systems have been proposed and are serving online right now: BigTable (Gruber 2006) in Google, Cassandra (Lakshman and Malik 2009) and Hive (Hive 2011) in Facebook, HBase (Apache 2009) in Streamy, PNUTS (Cooper, Ramakrishnan et al. 2008) in Yahoo! and many other systems. In order to merge with the Cloud computing platform, the data management systems should have high availability and fault tolerance, flexible scalability, and the ability to run in the

heterogeneous environment. Developers of existing systems choose different solutions to make their data management systems work well in the cloud depending on their different application scenarios. But storage is only the first step for enterprise applications, there is still a lot of work needed to be done in the data management systems such as searching performance, heterogeneous data clustering, weighting methods for data fusion. For example, from the storage part, index and data partition methods will be added and gain better performance in searching. From the data analysis part, future selection methods and weighting methods will be used for correlation analysis to make better use of the data.

Index is definitely an overwhelming strength of relational databases. No matter using relational database or not, researches have made their efforts to build index on Hadoop, some other researches prefer, Trojan index (Dittrich, Quiané-Ruiz et al. 2010), full text indexing (Lin, Ryaboy et al. 2011), HAIL(Dittrich, Quiané-Ruiz et al. 2012) and etc.(Jiang, Ooi et al. 2010, Richter, Quiané-Ruiz et al. 2012). These researchers have managed to add index to Hadoop. However, some of them could only handle full text search. For others, they've done much to optimize query time cost, but bandwidth cost during query process is still not well optimized. The full-text indexing research based on Lucene could inform the Hadoop execution engine in which compressed data blocks contain query terms of interest, and only those data blocks are decompressed and scanned (Dittrich, Quiané-Ruiz et al. 2012). Though this research manages to leverage a full-text index to optimize selection operations on text fields within records, this research is based on Lucene, which suits full-text searching more than unstructed data. Hadoop++ could injects Trojan indexes into Hadoop input splits at data loading time, but index could only be created per logical HDFS block rather than per physical replica (Dittrich, Quiané-Ruiz et al. 2010). So there comes HAIL (Dittrich, Quiané-Ruiz et al. 2012), namely Hadoop Aggressive Indexing Library, to fill this gap. It is an index system on Hadoop that manages to reduce high upfront indexing costs and support multiple sort orders. Another index LIAH could create clustered indexes on HDFS data blocks as a byproduct of executing MapReduce jobs, and has a very low indexing overhead, usually for the very first job. (Richter, Quiané-Ruiz et al. 2012) However, all these indices lack the flexibility to support multi-schema dataset, and none has involved map filter to reduce bandwidth cost.

Feature Selection is the process of selecting a subset of features that can properly

characterize the dataset. Correlation-based feature selection such as CFS(Hall 1999) and mutual information-based feature selection such as mRMR(Peng, Long et al. 2005) and MIFS(Battiti 1994) are the two widely used feature selection methods(Guyon and Elisseeff 2003). Their basic ideas are similar: good feature subsets contain features correlated with the class and uncorrelated with each other(Hall 1999). In these methods, they use statistics like mutual information to measure the dependencies among features and this idea is adopted in our method to determine the angels between axes. Kernel based PCA (Schölkopf, Smola et al. 1998) and Riemannian manifold learning(Lin and Zha 2007) methods are all very popular this days, but these methods are all focus on global optimizations on dataset which leads to low performance.

In enterprise application, especially in the online payment company, real time fraud detection in transactions are the key problems in their business. All transaction information and other social network information are all stored in graph manner. How to analysis the potential fraud transaction in such a big graph in a limited time is a big challenge. Current research are all focus on performance turning based on database technology(Parmar, Vaghela et al. 2015) or graph processing technology(Salihoglu and Widom 2013), but is not very suitable for large data processing.

1.3. Motivation

With the growth of the volumes of data sets and the rapid development of cloud technologies, efficient data storage, data modeling and data analysis methods are required to handle and cover the challenges with reliability. Especially with the idea of fraud detection in financial company, transaction records, offline records and social network records will be merged together. The more data about the user, the better potential fraud transactions to find in financial companies. Some software technologies will be involved to build a fraud detection platform to help the financial company to make the best use of data and find out the potential fraud transaction.

1.4. Research Aims

The research aim is to develop the concept of serials of software technologies for effectively sharing, querying, analysing multiple, large volume and heterogeneous datasets, which will stimulate participants to share knowledge and data, help to make the best and effective use of data and generate innovative ideas, and provide a basis for sharing intellectual property during the process. Currently a lot of the organizations have their own data management and knowledge mining system, but they all facing performance and scalability problems. For example, in eBay/Paypal, they will have several Terabytes transactions per day and all the fraud detection for online transaction will be finished in few minutes. How to store these large volumes of internal transaction data and external data for multiple purpose analysis in nearly real time becomes a great challenge for them.

In our research, since the data sets may come from different sources and with different formats, the storage subsystem should be able to deal with structured data from the database system and also the unstructured data from the internet and social networks. All the structured and unstructured data can be handled in the same way. The storage subsystem will also deal with the large volume of data sets, the date set needs to be stored separately for better scalability and need to be indexed for better performance. The data sets and the knowledge can be easily added into the system for better extendibility.

Since there are a lot of different kinds of data nodes and links in the fraud transaction prediction graph, the knowledge clustering subsystem should cluster based on both the graph attribute and graph structure consistency. Different nodes can be clustered together and the linkage can be dealt with as a weight automatically.

1.5. Research Contribution

- To develop a basis for data management system architecture based on heterogeneous graph, distributed storage and parallel processing

technologies.

- To develop a basis for knowledge finding and sharing system based on entropy based data clustering and attribute selection technologies.
- To develop a fraud detection prototype system which will use the proposed data management and knowledge finding method.

Most recently, many approaches and projects have been investigated as to how to build data management system. However, there is still no exact agreement and solution on how to store the data and knowledge, how to model the knowledge and how to analysis the knowledge and dataset.

This research will benefit the data management system, as it is flexible and adaptable. An indexed distributed heterogeneous graph storage system will be proposed, which will regard knowledge as a heterogeneous graph and store the dataset in an indexed distributed file system for better performance and scalability. A knowledge clustering algorithm will also be proposed, which can deal with the heterogeneous graph for different kinds of nodes and edges. Moreover, the data management system technology and knowledge finding algorithms will be used to build a fraud detection system for an online transaction company to evaluate all the proposed technologies.

1.6. Dissertation structure

CHAPTER 1 presents the introduction of this research. The technical background of data management system, problem statement, motivation, research aims, research contribution and the dissertation structure are described.

CHAPTER 2 presents the state of the art of this research, which gives an overview of the recent technological developments in distributed storage, parallel processing and graph based knowledge management.

CHAPTER 3 discussed the architecture of the proposed data management and knowledge sharing platform. Including the knowledge modeling, storage subsystem and knowledge clustering subsystem.

CHAPTER 4 introduces the proposed storage subsystem, including the philosophy of heterogeneous graph knowledge storage and index on the storage system to achieve better performance. Give the design and the implantation of the logical system and the

optimization for the physical implementation.

CHAPTER 5 introduces the proposed knowledge clustering subsystem, including the entropy based feature selection algorithm and non-Euclidean geometry weighting algorithm to deal with the heterogeneous knowledge graph analysis problems. The attribute generation and structure consistency in heterogeneous knowledge graph clustering are also be analyzed in this part.

CHAPTER 6 verifies the prototype of the data management and knowledge clustering system through a sample application.

CHAPTER 7 summarized the research and future work.

2. Literature Review - The State of the Art of Data Management Systems

This chapter presents the state of the art of data management systems. The analysis of different storage strategies, knowledge clustering and knowledge sharing strategies including their advantages and disadvantages. This chapter will be structured as follows:

2.1 Data management state of art

2.2 Distributed storage system state of art

2.3 Paralleled processing system state of art

2.4 Graph based knowledge management system state of art

2.1. Data management

Data management systems make use of computer hardware and software technologies to make a good and effective data collection, data storage, and data processing. The purpose of a data management system is to fully and effectively use data resource. With the development of computer and software technologies, data management systems have changed from file system, to a cloud based data management system.

A data management system is different from transactional database applications because data involved in analysis is rarely updated, so ACID (Atomicity, Consistency, Isolation and Durability) guarantees, in the transactional applications are not needed in most cases. Brewer rise the CAP theory(Brewer 2000), which means Consistency, Availability and Partition Tolerance cannot be achieved in the same time. In 2002, BASE is proposed to extend the CAP theory, BASE means Basically Available, Soft-state and Eventual consistency, which becomes the theoretical foundation of modern NoSQL (Not only Structure Query Language) technology.

RDBMS (Relational Database Management System) and EDW(Enterprise Data

Warehouse) are still widely used for data management(2013), Data analysis applications are often deployed on full distributed parallel databases, but with the increase of data scale, systems will have to scale vertically which costs a lot of money and time to get better performance. Distributed file systems which provide storage, fault tolerance, scalability, reliability and availability are also developing quickly.. Cloud-based data management systems provide a flexible and economical solution to horizontal scaling with commodity hardware, and the scaling of server resources can be transparent to the applications. In web data management applications, response time is one of the most important requirements aside that of scalability and fault tolerance. Big data is produced during the interaction between customers and the sites, and and the volume of such data is likely to continue to increase. Many companies, which supply social network services, have moved some of their applications to cloud-based data management systems because of this data explosion. Existing cloud-based data management systems utilise techniques such asBigTable, HBase, HyperTable, Hive and HadoopDB, (Abouzeid, Bajda-Pawlikowski et al. 2009) for analytical data management applications, whilst, for example,PNUTS and Cassandra are used for web data management.

Depending on how to store the data, there are two kinds of cloud based data management system: File System-based systems and DBMS-based systems. The typical File system based systems are BigTable(Gruber 2006), HBase(Apache 2009), HyperTable(hypertable 2009), Hive(Hive 2009) and Cassandra(cassandra 2013). HBase and HyperTable are open-source implementations of Google's BigTable's architecture, they are called the BigTable-like systems. Hive stores data in a master slave organized distributed file system. Cassandra uses file system directly as the peer-to-peer organized storage layer. The typical DBMS based system are SQL Azure(Microsoft 2012), PNUTS and Voldemort(Voldemort 2011). This DBMS based system all store data in a database, which is well designed for query optimization, index techniques, and it's easy for this kind of system to support SQL to the users. However, under this kind of architecture, data storage is limited, because all of the data storage optimization can only be executed on top of DBMS.

Generally speaking, File based systems can benefit from the wide use of

distributed processing models such as MapReduce in order to improve scalability, query execution performance, and fault tolerance in an heterogeneous environment.(Chi, Song et al. 2007) . How to execute SQL (Structured Query Language) and get performance gain from traditional database technology is still a big challenge and a lot of work existed on this area. For example, Hive (Hive 2011) made a great achievement to support part of SQL called HQL(Hive Query Language). DBMS-based systems can support SQL, but there is still a lot of work to do on scalability, fault tolerance, and support for semi-structured and unstructured data.

The data models of existing cloud-based data management systems are extremely different, there are two kinds of basic models which are widely used: the key-value data model and the simplified relational data model.

The key-value data model is a sparse, distributed, persistent multidimensional sorted map(Airoldi, Blei et al. 2008). The concept for a key-value pair comes from hash table in data structure, this model is very simple and flexible. In a key-value data model, rows and columns are the same as in a relational database, but it is not required for each row to have the same structure, or have the same number of columns. Different reading schema for each row during the process of data loading can be added(Jing, Haihong et al. 2011). For each key-value pair, the key is the identity for one pair and has to be unique, but a value can be a column family, which can be a list of columns. Each column can be different in data type, such as a time stamp or even a nested column family. The map relationship can be simply summarized as <row key, <column family, <column name, <timestamp, value >>>>. The relational data model is based on relational algebra, Hive is a typical system that uses a relational data model. All the data in Hive is organized as a table and each table is stored in a HDFS directory (Basu, Bilenko et al. 2004). Compared these two models, the key-value data model is simple and easy to implement. But this model can usually only support APIs instead of a uniform language like SQL, which supplies sophisticated DDL and DML operations. So there is still a lot of work to do to widen the application scope of the key-value model systems.

2.2. Distributed storage system

The traditional method for storing data persistently is to store data in file systems or relational databases. But in recent years, traditional methods are difficult to scale. A lot of researchers (Hasan, Anwar et al. 2005) have focused on using simpler storage systems that are easy to build and maintain. Google, Yahoo!, Amazon, Microsoft and some open source projects have already built some commercial and open source large scale data storage systems such as Bigtable, PNUTS, Dynamo, DRYAD and Hadoop.

2.2.1 Google: Bigtable

Google's Bigtable is the most popular distributed storage system for very large size scaling through thousands of commodity servers. Google search engine, Google finance, Orkut, Google Docs and Google earth all benefit from this approach. These Google services use Bigtable in different ways such as, batch-processing jobs and latency-sensitive serving of data to the end users. This approach works well for thousands of servers dealing with up to 100 TB of data.

Bigtable provides a simple data model to the end user and users can define a dynamic reading schema when they read the data, but it does not support a full relational database model. In particular, Bigtable is a distributed, simple, sparse and multidimensional sorted map. The map features a "key-value pair", and a timestamp to index the map. The end users usually need to serialize the format of structured, semi-structured and unstructured data into key-value pairs. A typical example would be the storage of a large group of web pages into a single table.

Bigtable uses a distributed file system, known as GFS (Google File System) (Ghemawat, Gobioff et al. 2003) to store and log data files. The Google SSTable, is used to store data files. Inside the GFS, they use Google's SSTable file format to store real Bigtable data. The SSTable is a persistent, sorted unchangeable map from key to value, and both the key and value are stored as byte strings. In order to achieve distributed storage, Bigtable relies on a distributed lock service, which is known as Chubby(Burrows 2006). Chubby consists of several active replicas,

and one of them will be elected as a master. The master server is alive when most of the other replicas are running and they can communicate with each other. Bigtable will use Chubby service for the election and unique of the master server, sorting the bootstrap location for Bigtable data and store Bigtable schema information and access control lists. If the Chubby system does not work for a while, the whole Bigtable system will broke.

During the runtime of Bigtable, it will allocate one master server and several slave servers, which can be dynamically added or deleted from the cluster system. If the workload is very high, more slave servers will be added to the cluster, or vice versa to reduce the running cost for the whole system. The master server is used to monitor, balance, the workload of each slave server and also manage the metadata of file blocks in GFS (Google File System). Each slave server manages a set of data blocks. When a read/write query comes, the master server will find the metadata of the file and file blocks, then the client will go directly to the related slave servers for reading and writing.

2.2.2 Yahoo: PNUTS/Sherpa

Unlike Google who use GFS (Google File System) to achieve scalability and distribution, Yahoo uses an extend database system for massive scaling of Yahoo!'s web application (Cooper, Ramakrishnan et al. 2008). The PNUTS system (renamed to Sherpa later) is a simple relational model where data is organized into tables of records with attributes. The purpose of the PNUTS system is to focus on the data serving web applications rather than complex queries, it has added some new data types to the traditional database such as blob, which allow user defined structure inside a blob. But PNUTS does still not support large unstructured data such as images, audio and videos. PNUTS has very flexible schema of tables, and can add attributes without stopping any query or update operation.

The PNUTS system can have several regions which might be geographically distributed. Each region contains a full complement of system components and copy of each table. So data tables in PNUTS will be partitioned into groups of rows as a block/tablets. They can be distributed and duplicated into many servers for better access performance. Each server will have thousands of block/tablets and

block/tablets can move between servers for load balancing.

The PNUTS system supports selection and projection from a single table. The design strategy for PNUTS is to optimize queries that read and write a single record or small group of records. PNUTS keeps the primary key and provides a multi get operation that can read multiple records in parallel by querying a set of primary keys. PNUTS has a router component that determines which storage contains the record being queried. The primary key space is separated by the router component into intervals, where each interval corresponds to one block/tablet. The router stores an interval mapping that defines the boundaries of each block/tablet and maps each block/tablet to a storage unit. PNUTS does not support for join, because the join operation is too expensive for such a massive scale system.

PNUTS system does not support log or archive data in a traditional database. Instead it uses pub/sub mechanism for redo logic if disk I/O failure happens when write or update happens. Because in web applications, it is very common to have to access one record at a time while different records may be accessed in different geographic locality, PNUTS provide per-record timeline consistency (Vogels 2009), which means all the replicas of a given record apply all updates to the record in the same order. In PNUTS one of the replicas is assigned as a master for the record. If any update happens in other replicas for that record, the update will be forward to the master. The way master replicas are assigned is that the replicas receive the majority of the write request for this particular record. Based on pre-record timeline consistency, the following API calls with different level of consistency guarantees in PNUTS can be achieved:

- Read-any: the lower latency and returns a possible stale version of the record.
- Read-critical (required version): returns a same or newer version of the record.
- Read-latest: returns the latest copy of the record that reflects all writes that have succeeded.
- Write: has the same ACID guarantees in transaction for a single write operation.
- Test and set Write (required version): write to the record that only the

present version is the required version.

PNUTS is designed to scale to use several worldwide replicas, failover and load balance is provided in the system. If any server failed, the system will automatically cope all the data from a replica to other working servers.

2.2.3 Amazon: Dynamo

Dynamo is designed for high scalability and reliability in Amazon. As a worldwide e-commerce platform. Amazon requires a high performance, reliabal and efficicient storage system. The system has to serve tens of millions users at peak time and from an architecture perspective the system has to support continuous growth in the future. To meet these needs, Amazon has developed a number of storage technologies such as: Dynamo System(DeCandia, Hastorun et al. 2007), Simple Storage Service (S3), SimpleDB and Relational Database Service (RDS).

The Dynamo system(DeCandia, Hastorun et al. 2007) is also a highly distributed key-value based data store. The design strategy for Dynamo is to provide well balanced availability, consistency, cost effectiveness and performance for single primary key access to the data store. Dynamo provides a simple primary key only interface to the end user. The query model for Dynamo is a simple read or write operation to a data item that is identified by the key. Dynamo also keeps state as a binary objects(blobs), which is identified by unique keys.

Dynamo uses a variant of consistent hashing mechanism (Karger, Lehman et al. 1997) to do the partition. Each node in Dynamo will be assigned a random value to present its position on the ring, which is the output range of hash function. When a new data is added, Dynamo will hash the key of the data and walk across the ring clockwise to find the first node which position is big than the hash value. In this case, each node becomes responsible for the region in the ring between it and its predecessor node on the ring. If any of the node failed in the ring, it only affects it's nearest neighbors node and not affect all the data sets.

The Dynamo system also provides support for data duplication; each data item will be replicated at N hosts in Dynamo. N is a configurable parameter for Dynamo. Each key is assigned to a coordinator node in Dynamo. The coordinator node will

continue to check the next N-1 successor node clockwise and duplicate the data. This results in a system where each node is responsible for the region of the ring between it and its N predecessor. The system is designed so that every node in the system can determine which nodes should be in this list for any particular key.

2.2.4 Microsoft: DRYAD

Dryad is a general-purpose distributed execution engine introduced by Microsoft for coarse-grain data-parallel applications(Isard, Budiu et al. 2007). Dryad is based on dataflow graph, which is a combination of computational jobs and communication jobs. Each of the vertices in the dataflow graph is executed and distributed on all available computers. It communicates through files, TCP pipes and shared memory FIFOs. In a Dryad system, the programmer can easily define the communication patterns by a directed acylic graph and can also define how the data communicates with other by files, TCP pipes or shared memory (FIFOs). The direct specification of the graph lets the developer easily extend and reuse traditional Unix utilities such as grep, sort and head.

Compared with MapReduce, Dryad allows graph vertices to use several inputs and outputs. In Dryad, a Dryad job is determined by its communication flow. Each vertex in a Dryad directed acyclic graph is a program and each edge represents data channels for communication. From the directed acyclic graph, a Dryad job is automatically mapped onto physical computing and communication runtime resources. There is a scheduler called job manager in Dryad system, which is responsible for the control decisions for the job, and can be deployed within the cluster or a user's workstation, which can access the cluster via the network. Thus the data transfers are not related to the scheduler avoiding bottlenecks in the whole system.

2.2.5 Open Source Projects

Most cloud data management systems proposed by companies such as Google, Yahoo and Microsoft are for internal use, but there is still a lot of open source projects have been built to implement the concept of these enterprise systems and

which are available for public use. The has made a big contribution to these open source systems.

Cassandra is the most famous high scalability, distributed key-value store (Lakshman and Malik 2009(Lakshman and Malik 2009). Cassandra is designed by Avinash Lakshman (one of the authors of Amazon's Dynamo), Prashant Malik (Facebook Engineer) and has been open sourced by Facebook in 2008. Cassandra is based on idea of Dynamo and Google's Bigtable, which provide eventually consistent and Column family data models to extend normal key-value system.

HyperTable is another famous high performance, scalability and distributed storage and processing system. The HyperTable is designed to work on large clusters. Each of the server in the cluster is not reliable and they may fail anytime. HyperTable provide a low-level API and query language(HQL) for the end users to create, modify and query the underlying tables. The HyperTable will try to process the data in parallel and also try to use the server that physically stored the data to computer.

CouchDB is document oriented database system. Documents are the basic unit of data. Each Documents in CouthDB is a combination of multiple fields, which can be string, numbers, lists or maps and will be given a unique ID. CouchDB provide a RESTful HTTP API to read and write database documents and all the documents will be processed in a MapReduce way. In CouchDB, the update for a document is lockless, which means if multiple update happend on the same document, a conflict error will be given from the system. Reopen the latest document version and update based on the latest version is the only choise.

2.3. Parallel Processing Systems

This section provides a survey of the state-of-the-art of the parallel processing systems, which have been proposed for the implementation of cloud data storage systems.

- MapReduce

With the growth of data volumes, database administrators have to find a way to process data within limited time. Usually there are two choices for this problem: scaling up or scaling out. Scaling up usually means using bigger and more powerful

computers to process the data, but the drawback of this choice is that bigger and more powerful computers are very expensive and have a physical limit when processing large amounts of data. Scaling out usually means distributed or partitioned data across multiple machines. In large data processing, scaling out is more extensible and economical than scaling up.

MapReduce is a programming model based on scaling out strategies for large data processing. MapReduce enables easy development of scalable parallel applications to process large amounts of data on large clusters of commodity machines.(Dean and Ghemawat 2008). With the MapReduce programming model, the programmers can easily run their jobs in parallel and do not need to pay attention to data distribution, scheduling and fault tolerance in distributed systems any more. The input and output of MapReduce will be key value pairs. Each of the servers in the cluster will automatically take a Map job to process part of the data and produce a set of key value pairs as an input to the next Reduce job. The Reduce job will automatically collect all the output of the Map job and merge them together. Typically just zero or one output value is produced per Reduce invocation (Yang, Dasdan et al. 2007).

The MapReduce framework is designed to run on large commodity server clusters instead of expensive high performance SMP (Symmetric Multiprocessing) or MPP (massively parallel processing) machines for better economical performance. For the storage system, MapReduce also uses each server's own local hard drives instead of centralized RAID-based SAN or NAS storage for the same reason. In the MapReduce framework, data will be automatically duplicated and the job will also be re-launched if any failure happens during the process. New servers can be easily plugged in at any time without a lot of administrationwork. There is no complicated backup, restore and recovery configurations like the ones that can be seen in many DBMS systems. The most important contribution for the MapReduce framework is that it provides an easy way for programmers to automatically parallel task executions. The developers only need to focus on the data processing rather than pay a lot of attention to memory management, file allocation, multi thread or network programming (Stonebraker 1986).

Hadoop is an open source Java project of the HDFS data storage model and MapReduce data processing model. A typical data process flow of Hadoop system

will follow the following sequence of actions:

First, the input files of the MapReduce program is split into m pieces and starts up many copies of the program on a cluster.

Second, one of the copies of the program is elected as a master copy and the rest will be assigned as workers by the master copy. The master copy will assign m map jobs and r reduce jobs to each of idle workers.

Third, a map worker will read the related data split as input and parses as key value pairs as output. Each map worker will buffer the output key value pair in memory.

Fourth, each map worker will write the buffered pairs to their local disk and the pair will be partitioned into r regions by the portioning function. The location of buffered pair on local disk will be passed back to the master. The master will forward this information to the reduce workers.

Fifth, when a reduce worker receives the information about the locations of the map worker output, they will read the buffered data form the map workers local disk and sort them based on the key value of the key value pairs.

Sixth, the reduce worker passes the key value pairs to the users reduce function. The output of the reduce function will be the final output file for this reduce partition.

Seventh, when all the map jobs and reduce jobs finished, the master program will wake up the user program and return back to the user code.

During the whole execution process, the master will get heart beat signals from all workers periodically. If there is anything wrong with the worker server, because the output is stored on local disk, the master will reset all the map jobs which is dispatched to that worker and find another idle worker to redo all the affected map jobs. The master will try to find the idle worker which contains the data that are needed in the map job to reduce the total transaction cost on the network.

From Dean and Ghemawat's (Dean and Ghemawat 2010) discussion on the MapReduce framework, they think that MapReduce is useful for dealing with data processing and data loading in a heterogeneous system with many different storage systems. MapReduce framework is very flexible for executing complicated functions and can support SQL directly. But they also point out 3 lessons when they

are using the MapReduce framework:

1. MapReduce users should take advantage of natural indices (such as timestamps in log file names) whenever possible.
2. Most MapReduce outputs should be left unmerged since there is no benefit of merging them if the next consumer is just another MapReduce program.
3. MapReduce users should avoid using inefficient textual formats.

Multiple datasets join operation is a big challenge in a MapReduce framework. This operation is related to the whole dataset rather than data blocks. Users can map and reduce one dataset and read data from another dataset block by block to do the join operation. Blanas et al. (Blanas, Patel et al. 2010) evaluated distributed join algorithms such as Repartition Join, Broadcast Join in MapReduce framework. Yang et al (Yang, Dasdan et al. 2007) proposed a Map-Reduce-Merge model to process multiple datasets. Yang et al. (Zhou, Cheng et al. 2009) also proposed improving the Map-Reduce-Merge framework by adding a new primitive called Traverse. This primitive can process index file entries recursively, select data partitions based on query conditions, and feed only selected partitions to other primitives.

The basic MapReduce framework requires that the whole output of each map job and reduce job to be materialized into a local file before the next stage start. The materialization step makes checkpoint/restart fault tolerance very simple and elegant to achieve, but also makes it difficult to process stream data. Condie et al. (Condie, Conway et al. 2010) proposed an enhanced architecture to pipeline intermediate data so that it can run MapReduce jobs continuously, accept new data as it arrives and analyze it immediately. It also allows MapReduce to be used in event monitoring and stream processing applications.

Recently, a lot of research has been done on applying the MapReduce framework to solving challenging data processing problems on large scale datasets in different domains. For example, Wang et al. (Wang, Wang et al. 2010) have presented the MapDupReducer system for detecting near duplicates over massive datasets. Surfer(Chen, Weng et al. 2010) and Pregel(Malewicz, Austern et al. 2010) systems have been designed to achieve efficient distributed processing of large scale graphs. Ricardo (Das, Sismanis et al. 2010) is a scalable platform for applying sophisticated statistical methods over huge data repositories.

2.4. Graph based knowledge management system

Clustering is a classical problem in data analysis and machine learning for multi-dimensional data analysis (Jain and Dubes 1988). Most of these algorithms are attribute based methods, but in graph based knowledge management systems, the link contains a lot of useful information for clustering. Some of the clustering methods (Chakrabarti, Kumar et al. 2006, Chi, Song et al. 2007, Kim and Han 2009, Bortner and Han 2010) will use network structure to solve community detection problems (Clauset, Newman et al. 2004, Backstrom, Huttenlocher et al. 2006, Leskovec, Lang et al. 2008). Recent work has attempted to overcome the heterogeneous network clustering problem (Sun, Yu et al. 2009)., but it is aimed at a star network schema, and is not applicable for general structure networks. Furthermore, it cannot easily be integrated with attribute information.

Recently, some studies (Basu, Bilenko et al. 2004, Shiga, Takigawa et al. 2007, Mei, Cai et al. 2008, Yang, Jin et al. 2009) have shown that by considering the link constraints in addition to the attributes, the clustering accuracy can be enhanced. However, most of these algorithms require that the network links, objects and their attributes are all homogeneous. A recent clustering method (Zhou, Cheng et al. 2009) integrates the network clustering process with categorical attributes by considering the latter as augmented objects, but the same methodology cannot be applied to numerical values. Some other algorithms (Shiga, Takigawa et al. 2007) can cluster objects with numerical attributes by combining the network clustering objective function with a numerical clustering objective function, but it is difficult to decide the weight to combine them, and cannot deal with the incomplete attributes properly. Zhang (Long, Zhang et al. 2007) provides a framework for clustering objects in relational networks with attributes. However, they studied a different clustering problem by clustering objects from different types separately, and did not study the interplay of importance of different link types and the clustering results. Probabilistic relational models, such as (Taskar, Segal et al. 2001), provide a way to model a rational database containing both attributes and links, but do not consider the scenario studied in this paper that clustering purposes could be different according to the specified attributes. Also, they cannot

handle the problem of incomplete attributes due to the discriminative nature of their methods.

There are several different philosophies on using the link information in addition to attributes to help the clustering in networks. First, in (Shiga, Takigawa et al. 2007, Zhou, Cheng et al. 2009), links are viewed to provide another angle of similarity measure between objects besides the attribute-based similarity measure, and the final clustering results are generated by combining the two angles. Second, In relational clustering (Long, Zhang et al. 2007) and probabilistic relational models (Taskar, Segal et al. 2001), every link is treated as equally important and the probability of a link appearance is modeled explicitly according to the cluster memberships of the two objects of the link, in a way of building mixture of block models (Airoldi, Blei et al. 2008). Third, in (Mei, Cai et al. 2008, Sun, Han et al. 2009), links are considered to provide additional information about the similarity between objects that are consistent with the attributes, and the final clustering result is a more smoothing version compared with the one merely using attributes. However, none of these views is able to model the fact that different relations should have different importance in determining the clustering process for a certain purpose. Our philosophy in modeling link consistency is more similar to the third line, that is, two objects linking together indicates a higher chance that they have similar cluster memberships. Moreover, associate each type of links with a different importance weight in measuring the consistency under a given clustering purpose, and thus each type of relation carries different strengths in passing the cluster membership between the linked objects.

Traditional graph clustering mainly focusses on homogeneous graphs of single-type nodes. Graph partitioning divides a graph into subgraphs by finding the best edge cuts of the graph. Several edge cut objectives, such as the average cut (Chan, Schlag, and Zien 1993), average association (Shi and Malik 2000), normalized cut (Shi and Malik 2000), and min-max cut (Ding et al. 2001), have been proposed. Various spectral algorithms have been developed for these objective functions (Chan, Schlag, and Zien 1993; Shi and Malik 2000; Ding et al. 2001). Multilevel methods have been used extensively for graph partitioning with the Kernighan-Lin objective, which attempts to minimize the cut in the graph while maintaining equal-sized clusters (Hendrickson and Leland 1995; Karypis and

Kumar 1998). (Yu, Yu, and Tresp 2005) proposes the graph-factorization clustering for soft graph partitioning, which seeks to construct a bipartite graph to approximate a given graph. (van Dongen 2000) proposes a graph clustering algorithm based on flow simulation. (Shental et al. 2004) formulates graph clustering as inferences in an undirected graphical model.

Clustering on a special case of a complex graphs, a bipartite graph consisting of two types of nodes, has been touched in the literature as co-clustering. (Dhillon 2001) proposes a spectral approach on a bi-partite graph. A generalized co-clustering framework is presented by (Banerjee et al. 2004a) wherein any Bregman divergence can be used in the objective function. Another special case of a complex graph is a k-partite graph consisting of multi-type nodes. (Long et al. 2006) proposes a framework of relation summary network to cluster k-partite graphs. (Gao et al. 2005) proposes an algorithm based on semi-definite programming for clustering star-structured k-partite graphs. Note that these approaches focus on special complex graphs with only heterogeneous links and cannot make use of homogeneous link information.

3. Design and Developing an Innovation Data Management and Knowledge Clustering Platform Architecture

This chapter presents the architecture of the data management and knowledge sharing system developed in this research. It describes; analysis of the system requirements, data set and knowledge modeling, the design of the storage subsystem and knowledge clustering subsystem, and the design of the knowledge sharing subsystem. The structure of this chapter is shown below:

3.1 The requirement analysis

3.2 Knowledge modeling

3.3 Architecture for the data management and knowledge sharing system

3.4 Discussion

3.1. The requirement analysis

In the data management and knowledge sharing system the following questions are posed:

1. How to manage different datasets from different sources?
2. How to manage knowledge via the dataset?
3. How to make better use of the data set and knowledge?
4. How to bring about sharing of knowledge and data?

Within question 1, The following issues are addressed:

- The source data set may exist in different formats; it may be structured, semi-structured and unstructured data.

In most knowledge management systems, the datasets come from individuals or organizations. Some of these datasets will be well structured and based on the requirements of the originator, but it will not necessarily be very suitable for analysis and processing around a different requirement. Indeed, some datasets

will be semi-structured and even unstructured, such as web data and social network datasets. The key is to design a storage and access system, which will work in the same way for all different types of dataset irrespective of their source and format.

- The sematic description must be capable of being easily added in the data set. Users can easily understand the sematic meaning of data set.

In any data management system, it is very difficult to define a proper schema for a data set and never change it. In different applications, different schemas for the same dataset may be focused, and it is do needed to add more sematic meanings to the dataset during the process of analysis. For example, when dealing with a customer referral system, the address will be regarded as a whole sentence but when dealing with a logistical application, the address are needed to be divided into countries, states, streets etc., in order to quantify the data. Within the storage system, one dataset may have several schemas depending on the application.

- Large volumes of data need to be added and managed, so the scalability and performance of the system must be well designed.

With an increasing number of datasets within a data management system, the scalability, performance, and the processing of the data have to be carefully handled. Distributed storage and processing is a potential solution for this problem. In order to get better performance, some index can be added in the distributed storage and process system. The index needs to be easily scaled. The index are needed to be add and update based on graph schema, dataset schema, and the dataset. The index also needs to be processed in parallel for better performance. The idea of index and delayed We will be imported when load the dataset into the knowledge management system.

In question 2, The following issues are addressed:

- The knowledge needs to be abstracted from the data set.

Knowledge is an abstract idea-based information in the data set. Knowledge will come from multiple data sets and sometimes may not be linked directly with a particular data set, or multiple data sets. Knowledge will have its own attributes and linkage. Knowledge is usually defined by certain properties and there may also be linkages between different parts of the knowledge dataset.

- Knowledge needs to be modeled in a certain way.

It is proposed that a heterogeneous graph be used to represent knowledge. (Diati and Hoede 2008) in Netherlands initiated knowledge graph theory. The initial idea was to use a graph as a presentation of the contents of medical and sociological texts. In order to describe the different sematic meanings of "knowledge", a particular knowledge will be ragarded as a knowledge node type, atomic knowledge as an instance of a knowledge type node. For example, technical publications in a particular field can be regarded as a "knowledge node type"; a single publication in this field can be an instance of the node type. The "edge" is defined as the linkage between the knowledge nodes. For example, papers written by the same author. Thus, a heterogeneous graph will be built for knowledge management problems. The idea of a knowledge schema will be regard to describe the heterogeneous graph in the knowledge management system.

- The linkage between knowledge and data set needs to be dynamic without data duplication.

Knowledge is an abstract idea from a data set, and different elements of knowledge may come from the same data set. Since the volume of data sets are very high, if the data set is duplicated to represent multiple knowledge, the data volume will increase dramatically. It is proposed that a hype link between knowledge and data set without real data duplication to avoid the rapid increase of data.

In question 3, The following questions will be counted:

- How can the different kinds of knowledge be used together?

In knowledge a management system, how to make a good use of knowledge is a key challenge. There are so many knowledge types in the heterogeneous knowledge graph, the traditional clustering and classification methods will face some problem. Each knowledge type has its own properties. If all the types are analyzed together, each knowledge type will have a lot of missing data in the universe set; if each types are analyzed separately, the connection or weight between them is hard to count. We will try to find out the efficient feature selection method in heterogeneous graph analysis, which will use entropy-based feature selection algorithms to deal with it. Find out the key features in the prediction problems to

- The linkage between knowledge nodes can be used together

In knowledge heterogeneous knowledge graph, there are different kinds of linkage between nodes, different types of linkage will have different effect on a real analysis problem, which linkage is valuable and how to count the weight of linkage are key questions in making good use of linkage. An entropy based non-Euclidean geometry method will be used to help count the weight of the links.

In question 4, The following questions will be counted:

- How to provide basis for sharing intellectual property during the process

Intellectual property protection is a key problem in data management system. Authority will be added both on data set and data schema. All the data set will be encrypted and the access of data set and schema will be granted. An encryption and grant center for the whole data management system is needed to try to protect the IP.

- How to encourage the data sharing during the process

Encourage more and more organizations and individuals to join the data management system is a key to make the system work. Some serious game ideas will be involved into the data management system. A ranking system will also be involved into the data management system, the more data you contribute, the higher rank you can get, and the more data you can see and use in your future analysis.

3.2. Knowledge Modeling

Knowledge graph theory is a kind of new viewpoint, which is used to describe human language, while focusing more on the semantics than the syntactic aspects. Ontological aspects of knowledge graphs are discussed by comparing with important other kinds of representations. It is expounded that knowledge graphs have advantages, which are stronger ability to express, to depict deeper semantic layers, to use a minimum relation set and to imitate the cognition course of mankind etc. The formal description of knowledge graph will be added in data management in 3.2.1.

3.2.1 Knowledge Graphs in data set management

In data set management, the composition of knowledge graph is including concept (raw data set, data schema, data tokens and knowledge types) and relationship (binary and multivariate relation).

(1) Raw Data Set

A data set is the row data files coming from different organizations and individuals. All the data set coming from different sources may be in different formats, some are structured from a database, some may be unstructured from websites. It is necessary to do some of data cleaning and data conversion to all the data sets in order to make it a raw data set.

Definition 3.1 A Raw Data Set is a data set coming from different sources. The raw data set is validated from different source and unformatted. It is the data set input to processing.

(2) Data Schema

A data schema is the format description of a Raw Data Set, it describes the structure of a Raw Data Set. The data schema provider might be the same with the raw data provider, might be somebody else. One Raw data set might have more than one data schema.

Definition 3.2 A Data Schema is the description file for a Raw Data Set, it contains the structure information of a Raw data set. One Raw Data Set may have more than one data schema.

(3) Data Tokens

A data token in a segment of single line of raw data with a schema, the schema describes a structure of the raw data set and a single line of raw data will be separate into different segments. Each segment of data will be regard as a data token.

Definition 3.3 A Data Token is a segment of raw data, which is determined by the raw data and data schema.

(4) Knowledge Types

According to the viewpoint of subjectivism, different persons may describe experiences of the real world by different data tokens. If an experience can be

shared by most persons, or can be shared by all persons, it will be regarded as a knowledge types. A Knowledge Type refers to a logical common idea from a combination of several Data Tokens which from the same or different Raw Data Sets. Knowledge type will have it's own properties, which is also a particular knowledge type. For example, knowledge type "Book" will have properties like "Book Name", "Author" and "Publisher", which are also knowledge types.

Definition 3.4 A Knowledge Type is a common logical idea from a combination of several Data Tokens. The common logical idea represents certain knowledge.

To describe the real world, distinguish the relationships between knowledge types is needed. In knowledge graph theory, the most important principle is to use a very limited set of relationships. These relation types are required to be basic. The more independent these types are semantically, the better. It should not be possible to deduce one relation type from others. The meaning of these relations can be described through considering the relationship with the real world. These basic relation types can be used for establishing more complex concepts and relations. How to choose these basic relations in knowledge graph theory?

Definition 3.5 If a knowledge type conceptbelongs to another knowledge type, the relationship between the two knowledge types is an ISA-relation.

Definition 3.6 If a knowledge type concept is part of another knowledge type, the relationship between the two knowledge types is an HSA-relation.

Definition 3.7 A knowledge type in a knowledge graph has an incoming SKO-relation from another knowledge type if it is informational dependent on that knowledge type.

The basic goal of the knowledge graph project is to use a limited number of relation types. Only if the relation types are not enough to express something, a new relation type will be forced to add. To express the dependency relation, a new relation type must be considered, which corresponds to mappings. Though in natural language there are many words to express mapping, one relation type will be choose to express this relation, which is called SKO (Skolem)-relation. In natural language words like "depends on" are used. The meaning of the SKO-relation is based on the concept of informational dependency.

3.3. Architecture for the data management and knowledge clustering system

The data management and knowledge sharing system will be divided into Data Set Storage System, Heterogeneous Knowledge Graph Storage System, Heterogeneous Knowledge Graph Analysis System and a Data Presentation System.

After data cleaning and data conversion, data is loaded from the data source to the data set storage system. In the data set storage system, add the data schema is added to describe the format of the data set, and build the index to the data set based on the schema. Then the data set and schema will be used to generate the heterogeneous knowledge graph. With data management interface, the knowledge graph will be stored into the heterogeneous knowledge graph storage system. After the heterogeneous knowledge graph is generated, the entropy based feature selection algorithm and non-Euclidean geometry weighting algorithm will be used to analysis the heterogeneous knowledge graph. The analysis is handled in the data presentation system to show the result to the end users. The whole system architecture is show in Fig 2.

With this architecture, a distributed index in the data set storage system will be involved, which enhance the search performance, especially for real time fraud detection. A heterogeneous graph will also be involved to model complex business models in real data analysis problems. With the help of Non-Euclidean Geometry weighting algorithm and entropy based future selection algorithm, the features will be limited for fraud detection, which enhance the detection performance.

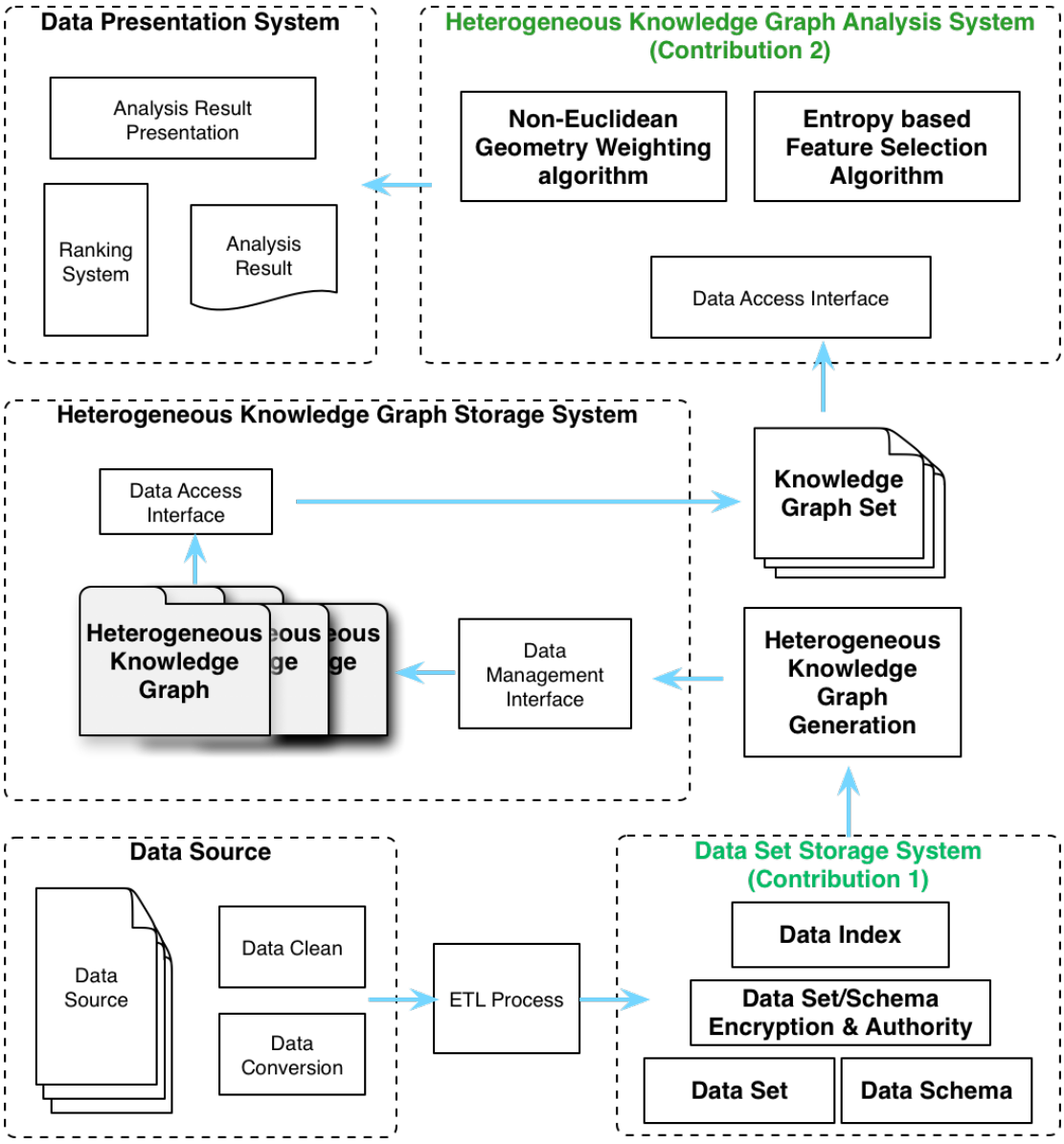


Fig 2. System Architecture for the data management and knowledge sharing system

3.3.1. Architecture for the storage subsystem

The storage subsystem contains data set storage system and heterogeneous knowledge graph storage system in fig 2. Based on concept of the HDFS (Hadoop File System), a heterogeneous graph and index will generated to guild our knowledge storage subsystem.

HDFS is an open source distributed file storage system based on GFS (Google File System). The key benefit of HDFS is schema on reading compared with RDBMS (Relational Database Management System), data is simply copied to the file store without transformation and a SerDe(Serializer/Deserializer) is applied during

read time to extract the required columns, which helps the load to be fast and more flexibility and agility. But the HDFS doesn't support for heterogeneous graph storage and there is no index support to optimize the reading performance. The concept of heterogeneous graph will be added and distribute it into the HDFS system, add index for both data set and heterogeneous graph to improve the HDFS to suitable for data management and knowledge sharing. The architecture for the storage subsystem is show in Fig 3.

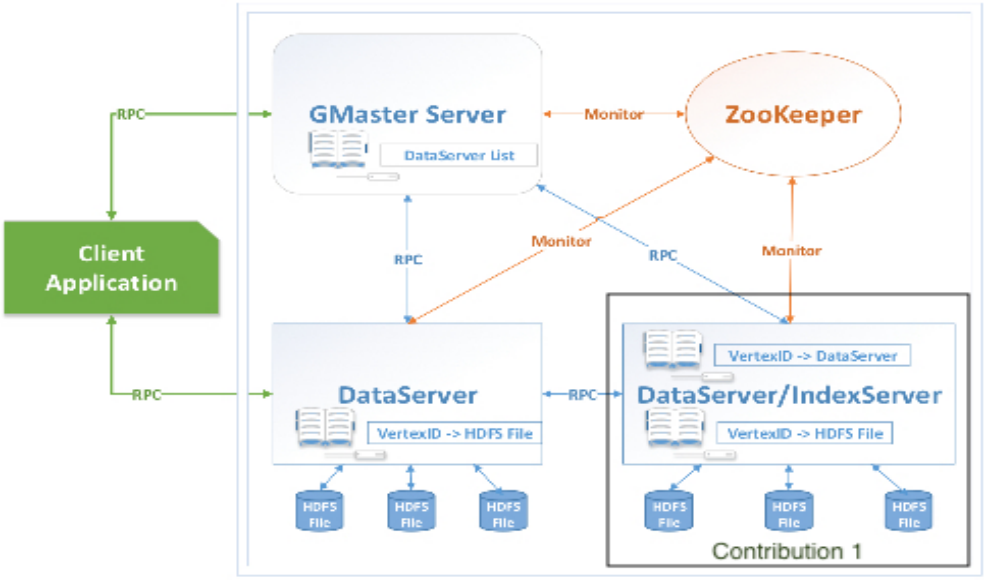


Fig 3. System Architecture for the data management and knowledge sharing storage subsystem

3.3.2. Architecture for the knowledge clustering subsystem

The knowledge clustering subsystem contains the heterogeneous knowledge graph analysis system in Fig 2. The key problems with heterogeneous knowledge graph analysis is how to deal with the attribute similarity and structural similarity. In the heterogeneous graph, different nodes have their own attributes. Each attribute is associated with the object and the attributes are not independent with each other, when combine the attributes together, there is still a lot of the missing values in the data set. All these problems will be handled in the attribute similarity.

In a heterogeneous graph dealing with the linkage between different nodes is called structural similarity. The linkages between different linkages need to be properly weighted and affect the final clustering. In proposed system Entropy based feature selection algorithms and non-Euclidean geometry weighting algorithms deal with these problems.

Entropy based feature selection algorithms: Given two continuous random variables x and y, their mutual information is defined as below(Cover and Thomas 2012):

$$I(X;Y) = \int_{\Omega y} \int_{\Omega x} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} dx\, dy \qquad \text{Eq. 1}$$

In equation 1, p(x) and p(y) are marginal probability density functions of X and Y respectively and p(x,y) is the joint probability density function of X and Y. In a data management and knowledge sharing system, the relevance of one explanatory attribute with the response attribute/object can be described by the mutual information value of these two attributes The larger mutual information value shows that these two attributes are more relevant. The redundancy among explanatory attributes also can be evaluated by mutual information value. However, the larger mutual information value is, the more redundant these explanatory attributes are.

Non-Euclidean geometry weighting algorithms: As previously discussed many of the attributes within datasets might have certain relevance with each other, and usually it is hard to find many attributes that are completely independent of each other. Thus, a non-Euclidean geometry weighting (NeGW) measurement is proposed for the case based reasoning method to deal with the relevance between attributes is considered in the distance equation. The key idea in NeGW is that mutual information is be used to define the angles between attributes axes. The mutual information shows the statistic linkage between the attributes, if the two attributes are in depend, the mutual information between them should be 0; the two attributes' axes will be orthogonal. The higher for the mutual information is, the more dependence they have, the two attributes' axes should be closer to be paralleled. So the mutual information can be used as the angles between attributes axes.

4. Design and Implement of Data Storage Subsystem

This chapter introduces the design and implementation storage strategies of the dataset management and knowledge sharing system,

4.1. Data Management storage subsystem analysis

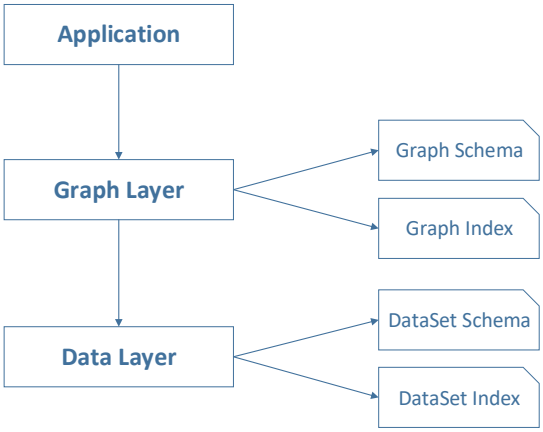


Fig 4. Two-Layer Multi-Schema Data Model

Since our proposed system is designed to manage knowledge as heterogeneous graphs, both the graphs' metadata (e.g. vertices and edges) and underlying datasets (which contain the real data) need to be stored and maintained. Fig. 4 demonstrates the logical view of the proposed data model.

As can be seen from Fig. 4, the graph layer and data layer are stored and managed independently. The graph layer represents the data as vertices and stores their relationships as edges; and the data layer manages all related datasets. Note that instead of storing data directly in the vertices (or edges), the graph uses pointers linking vertices to the datasets. Specifically, the relationship between graph and dataset is many-to-one, which means a single vertex can only link to one record of data, but one record can link to multiple vertices. Based on different angles to represent the data, multiple graphs (with different structures) can be

built on the same datasets to improve the possibility of data reuse.

To support data with different types, both layers are equipped with schemas and indices. The graph layer uses schemas to store attribute lists for different types of vertices (and edges), while data layer uses schemas to parse datasets with different data types. Next the details design of graph layer and data layer will be used respectively.

4.2. Indexed distributed graph-based data management and knowledge sharing storage subsystem design

The distributed storage system is based on the Google distributed file system GFS (Chang, Dean et al. 2008). It contains four parts: data nodes, which contains the dataset; data schema nodes and graph knowledge schema nodes, which contains the dataset schema and graph schema; index nodes, which contains the index for dataset; graph knowledge nodes, which contains the knowledge graph. Fig 5 shows the architecture of the proposed multi indexed knowledge management storage system.

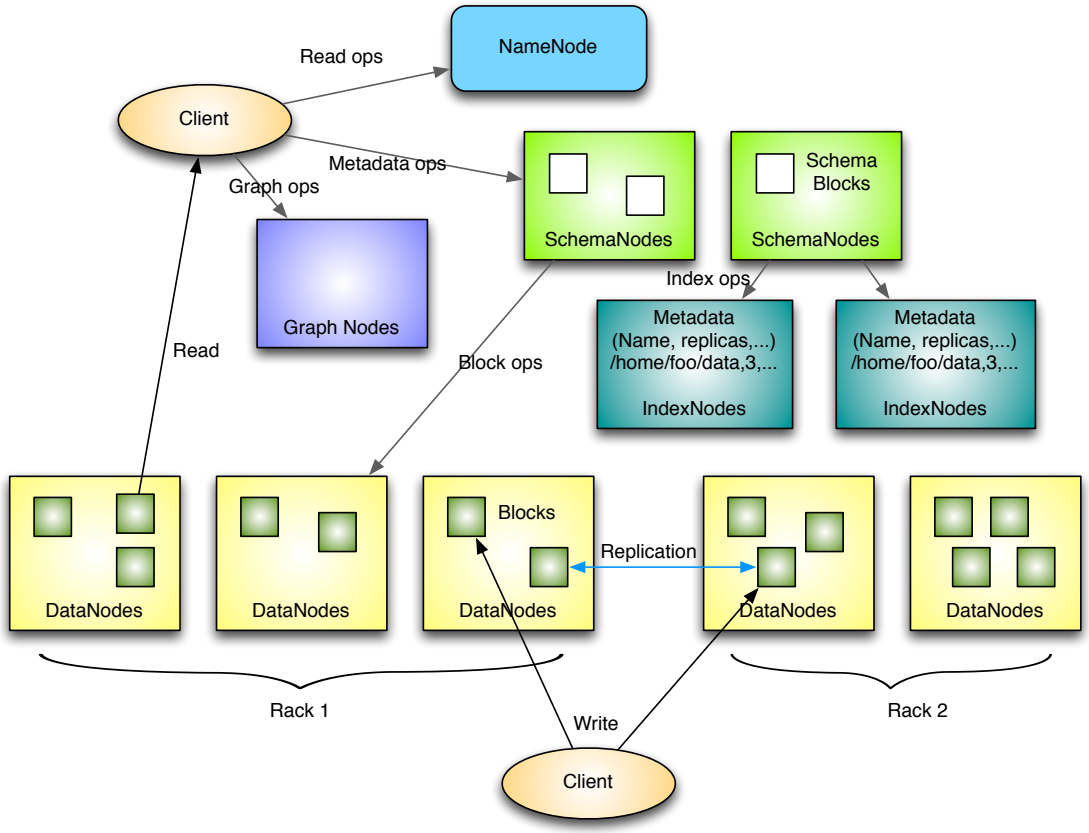


Fig 5. Architecture of Multi-Indexed Storage System in Data Management and Knowledge Sharing

The dataset is stored as a pure row data in the distributed file system. As an open source software framework that supports data intensive distributed applications, Hadoop (Dean and Ghemawat 2008) is designed for big data processing, supports the running of applications on large clusters of commodity hardware. An index features has been used in the Hadoop system to achieve the knowledge management system. Regard the dataset, no matter whether it is a structured data or unstructured data, as files, which will be stored in the Hadoop system as row files. During the processing, delay the read loading data of the dataset has been used for better performance.

The data schema is the description about dataset and the metadata of data index. One dataset may have more than one data schema. Each schema contains the self-described dataset structure and the metadata of the index. Since each row of data in the dataset has its own sematic meaning, the self-described dataset structure will present the position-sensitive or separate-sensitive meaning of the row data. The metadata of the index includes the availability of the index in each sematic meaning section of the dataset, the name, replicas, position of the index

block. With the metadata of the index, it will be easily find where the index block is. The data schema can be added during import or first load the dataset, but more schemas can also be added after that.

The data index gives the metadata of the data blocks. With the index system, the exact position of the data blocks can be located, so the minimal numbers of jobs canbe started to read the dataset, which reduces the total running jobs in the distributed storage and processing system. The data index can be built and updated dynamically, based on the availability of the resource and the statistics of the query, how many indexes are needed will be choose and update it during the "off peak" time of the system.

The knowledge graph schema is very similar to the data schema. It is the description about the knowledge heterogeneous graph and the metadata of the knowledge index. The information of the knowledge graph has three parts: node type, node and edge. Each category of the knowledge will be described as a node type; the instance of knowledge will be regarded as a knowledge node; and the edge will be the linkage of the two knowledge instances. The metadata of the knowledge index contains the availability of the knowledge index and the block information of the knowledge index.

The knowledge graph node stores the knowledge graph. Knowledge will be regarded as a heterogeneous graph; the knowledge graph node contains the knowledge atomic item and the linkage between knowledge atomic items. In order to avoid the duplication of data set, the knowledge node will only contain one or several pointers, which point to the dataset.

4.2.1 Details of logical design

Name Node: The name node in our multi-indexed distributed knowledge management storage system is nearly the same as GFS name node. In GFS, the name node contains the metadata of the data node, but in our multi-indexed distributed knowledge management storage system, the index node also contains the block metadata. That means when "fall over" happens, update both the name node and index node are needed. In order to reduce the synchronization cost, a hash table of the logic block metadata and physical metadata in the name node has

been maintained, the index node only contains the logic block metadata and if “fall over” happens, there is no need to update the index node.

Data Schema Node and Data Index Node: The data schema contains the semantic description of the data set. One dataset can have more than one schema. The schema is the description of the sematic meaning of the data set. It shows the properties of the data set, the metadata of the properties, the availability of index for each property, and the metadata information of index for each property. A typical data schema file will contain six parts: dataset name, dataset description, property names, property description, property separation metadata and property index metadata. For the property separation metadata, it will support the separators, allies, the prefix and postfix, and the combination and iteration of the three. For the property index metadata, if the index is available, it will point to the data index block; if it is not available, it will point to the dataset block. With the data schema, the records (row data) of the raw data set can be divided into a couple of columns, and meaningful data can be extracted from every column. To improve the read flexibility and data reuse, the data layer uses dynamic binding as well. Specifically, the data pointers of vertices from the graph layer will bind to a data schema in the runtime, and the data schema is responsible for parsing data for the vertices. The advantages of this binding can be seen in two aspects. 1) Data schemas with different column descriptions can be built on the same dataset. So that one data set can meet the needs of different applications, which increases the possibility of data reuse. 2) The pointer from the vertex to the data schema can be changed during runtime. By modifying the identity of the data schema stored in the vertex, the pointers of the vertex can switch to another data schema so that the data read from the vertex will be adjusted without modification to the read dataset.

The data index contains the index information for each property; the index is full text index for the property. The value of the full text index is the data blocks for the key. The data index is stored as a B+ tree for better search performance.

Knowledge graph schema node: Knowledge derived from data can be interconnected and can interact with other data, forming numerous, large, interconnected and sophisticated networks. the interaction has been regarded as a heterogeneous graph. In our knowledge storage system, the graph schema and graph index are also added into the system.

The knowledge graph schema contains the sematic description of the knowledge. It includes details of which data schema will be used to support the knowledge, the property of the knowledge, the linkage between the knowledge data sets, and the linkage between the knowledge property and the dataset property. A typical knowledge schema file will contain eight parts: knowledge names, knowledge description, proper names, property description, property metadata, linkage name, linkage description and linkage metadata. For the property metadata, it will contain the linkage between the knowledge property and the data schema property. For the linkage metadata, it will contain information as to which condition the node will have edge between and the direction of the edge. The graph schema will be identified by a unique schema_id, and stores the attribute description as key-value pairs, in which the key is attribute name and the value is associated data type. A sample of graph schema can be explained as follows (in XML format):

```
<graph_schema>
    <id>gs_001</id>
    <attribute_collection>
        <attribute name=”id” type=”Int32” />
        <attribute name=”name” type=”Char(20)” />
    </attribute_collection>
</graph_schema>
```

The knowledge graph node contains details of the knowledge schema used, the knowledge atomic item, and the linkage between them. In order to avoid duplication of datasets, only the atomic item id， the node/edge type, and serious pointers, (which point to the related info of row data in each related dataset) are stored. There is no data duplication in the knowledge graph system. If there is a query about the graph id, it goes to the dataset directly. If the query is not, based on the knowledge graph schema, then it is transferred to the query to a serious of data schema query. Since most of the knowledge graph is very sparse, adjacency list has bene used to store the knowledge graph. Specifically, for each node, it saves all the outgoing edges from the node to form a list. The advantage of this structure is its simplicity, efficiency for out-neighbor queries and less occupation of space (compared with adjacency-matrix).

The following information describes the typical structure for a vertex (in XML format):

```
<vertex>
      <id>v_0001</id>
      <edge_list>
            <edge id=”e_1827” target_vertex_id=”v_2847”/>
      </edge_list>
      <schema_id>gs_001</schema_id>
      <graph_id>g_172</graph_id>
      <pointer_list>
            <pointer attribute_name=”id”>
                        <target dataset_id=”ds_227”
                        data_schema_id=”dss_285” offset=”167”
                        />
            </pointer>
      </pointer_list>
</vertex>
```

This design gives vertices ability to store and represent diverse types of data. Instead of storing any “real” data in the structure, the vertices use schema_id and pointers to bind data in the runtime from the dataset. Specifically, if it is needed to retrieve data from a vertex, at first the vertex needs to read the graph schema by schema_id. Then for every attribute, it follows the pointer to read data from the dataset. The advantage of this process is improved flexibility and data reuse. Since the data binding happens in the runtime, the data that can be retrieved, and can be changed dynamically by adjusting schema and pointers. For example, if the user wants to hide some data from the graph, he/she can simply change the attribute list of a certain graph schema, and all the associated vertices will be changed accordingly.

Query Process Analysis: Fig 6 shows a query process in multi-indexed storage system of knowledge management process. The HDFS (Hadoop Distributed File System) Client first sends an open request to the distributed file system (step 1 in fig. 2), then the request goes to the Name Node (step 2 in fig. 2). Depending on the open request sent to the open a dataset or a graph, the name node returns the proper schema node based on the file name/graph name.

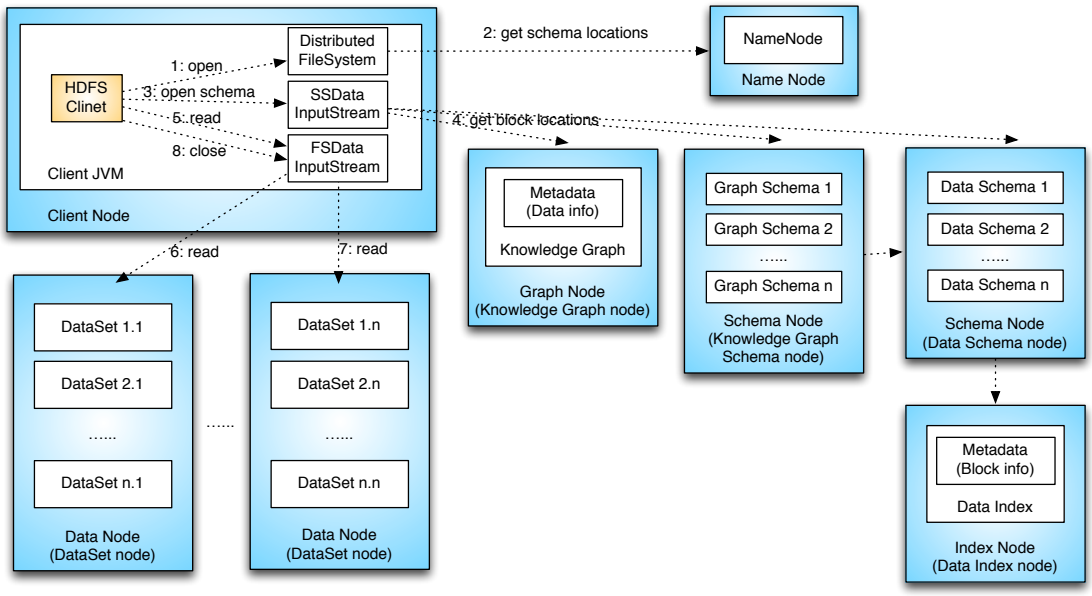


Fig 6. Query Process in Multi-Indexed Storage System in Data Management and Knowledge Sharing

The HDFS Client then sends the query to the schema stream (SSData)(step 3 in fig. 2). The SSData queries the schema node for the block location of the data (step 4 in fig. 2). If it is a query for a graph, the graph schema node will follow the graph schema description; find the data schema node, then goes to the data index to find the exact data block info. Or it will find the block information directly if it is a query of graph node id. The data block information will then be returned to the HDFS Client.

After the HDFS Client gets the data block information, it will send the query to file stream (FSData)(step 5 in fig. 2). The FSData then goes to the exact data block to read all of the dataset it needs (step 6, 7 in fig. 2).

4.3. Optimizations for the storage subsystem

4.3.1. Pure Distributed Design

- Reduce network transferring:

  From the logical design, the storage system has been divided into six logical parts: namely node, data node, data schema node, data index node, knowledge graph schema node and knowledge graph index node. But if all types of the node are stored separately in the system, the Task-Tracker needs

to load data max 3 times (query on the dataset) from the data schema node, data index node and data node; or 4 times (query on the graph) from the knowledge graph schema node, data schema node, data index node and data node. As we all know, transferring data in a local network system is a resource intensive job. The optimize for the performance of the storage system has been accomplished in physical design.

In our prototype system, most of the data schema files are founded to less than 1Mb, the size of knowledge schema files are nearly the same. That means if supposed to have less than 1000 schema files, the total size of the schema files is about 1Gb, which can be duplicated in all the data nodes, which reduces the cost of loading data schema and knowledge schema for the Task-Tracker. But in some large system, they may have more schema files, in this system, separation of the data nodes into several groups is suggested, each group containing part of the schema files.

For the data index, the index of each "column" in the schema has been regarded as a file, the index file will be regarded as a general file stored in the data node in the GFS system. The knowledge index will be processed in the same way as data index.

- Delayed data accessing:

  In a distributed system, load data and transfer data from a data node is resource intensive job, try to delay the reading of the dataset by better use of the index system. In DFS, if join, merge and diff is needed, the dataset have to be loaded first and then do the operation, which will lead to the cost of transferring data, unzip the data and a lot of related costs. But with the help of data index node, join, merge and diff operations can be done without loading the dataset. After the index of the particular data block has been founded, the join, merge and diff operations can be done on the index level rather than the data set level. Also, the same for knowledge graph node.

- When and how to build the index:

  In system design, even if there is no index, the whole system still can run, the schema node will contain all the data nodes for the dataset. That mean first when to build the index can be choose in need. In a knowledge management system, the overload of the whole system is not even, most of

the nighttime; the system overload will drop to the bottom. The index can be built when system overload is low because of the high cost of building the index. Second, how much index to build can be decided in need. As we all know, index will cost large volumes of disk space. How much index to be built and which indexes are most valuable are typical questions in the index system. Ideally, the more queries we have in the index is more valuable. In the prototype system, two thresholds have been added for the index system. First is the priority, how many times the index is queried by applications has been counted. The higher priority they have, the more chance the index on this property will be built. Second is max-size, which limits the maximum size of the index system. If the max size is reached, some indexes are deleted. Performance Analysis

Distribution issues: the whole multi-indexed storage system is based on the Hadoop distributed file system. Two new types of node are added, the schema node and the index node. They act very similarly with the data node; they all need to report the heart beat to the name node. The Master/Slave architecture is kept for the whole system. Different to some Lucene based index systems; the index and schema is separated from the original dataset for better scalability and control. Thus, the volume of the index can be simply limited, also based the statistical information of the query, dynamically change of index as needed and separate the update of index with the update of dataset.

Performance of execution: The purpose of adding an index to the system is to reduce the total jobs running on the knowledge management system. The exact block is located before map-reduce job started. The real loading of the dataset is also delayed. A lot of processing such as set join, merge and diff can be done at the index level [8]. Compared to the non-indexed Hadoop system, we increase the work of building the index and find the block info in the schema-index system. As described before, the building of the index is a separate job now and it can be done during the off-peak time of the system. The finding of the index will add data loading and process from schema node and index value before map-reduce started, which will add the traffic of the internal network of Hadoop system. But the size of the schema and the index

need to be load is small. If hit in the index, a lot of jobs can be reduced for map processing.

Performance of building the index: Add a default data schema to all the dataset, which marked as no attribute and all the blocks as index value. If an existing data schema with no index has been queried for a particular attribute, all the blocks of the dataset as its value will be added. Through these, the index system can be work even if there is no index. So, the building of index and based on the statistic info is delayed and build the index dynamically.

4.3.2. Semi-Distributed Design

Since the vertices are stored as HDFS files on distributed data servers, there are needs for the system to locate the data server and read the data for a certain vertex. To this end, a two-layer index structure is adopted. Specifically, the two layers are:

Global Index: mapping vertex_id to the address of data server that stored that vertex;

Local Index: mapping vertex_id to the path of HDFS files containing that vertex.

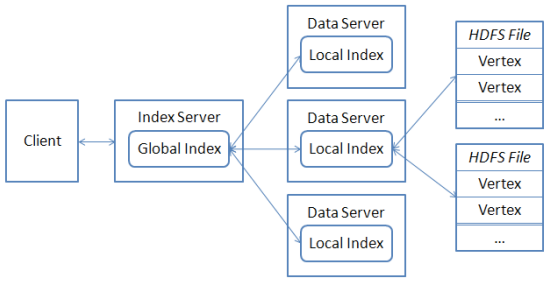


Fig 7. Two-Layer Index Structure

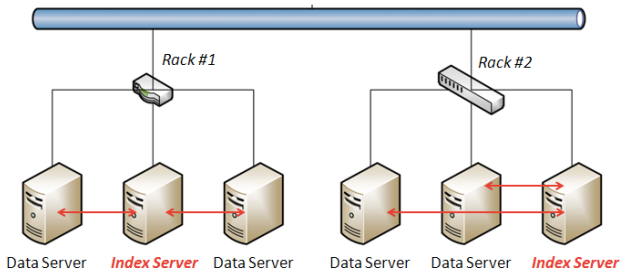


Fig 8. Network Topology of Index Server with 2 Racks

Fig. 7 depicts the structure of the graph index. Both layers use B+-tree to map the

vertex ID to the destination. If the client wants to read a vertex, at first it needs to use the global index to get the address of the data server that store the vertex, then it will contact the data server and that server will be responsible for querying its local index, reading the corresponding HDFS file and finally extracting the data. To ensure the index structure works, both layers of indexes are updated automatically when a vertex is inserted, migrated or deleted.

- Index Placement:

  With the 2-layer index structure, a question to ask is where to store those index instances. Obviously, the local index is built and stored on every data server, managing the vertices stored on that server. However, for the global index, it is tricky to select the right location to store. A natural way is to store the global index on the master server, but this will brings significant drawbacks as discussed before. Besides, since the size of global indexes is much larger than that of schemas, it is ineffective to store them on Zookeeper, which lacks support for large files. To tackle this problem, the Index Server is introduced to store global indexes, and provide query services to other data servers.

  The index servers are selected from data servers. In other words, apart from storing vertices, some data servers will store global indexes as well. To select the index server, there are two criteria: 1) the network delay between index server and normal servers should be as low as possible; 2) the workload of index server should be as low as possible. Under such considerations, an effective index selection strategy is designed, which can be depicted as follows: For distributed environment with multiple racks, one data server is selected from every rack; and the index server only provides services to data servers in the same rack. The data server with the lowest workload will be selected as the index server. Once assigned, the index server checks its workload regularly. If it is larger than a threshold, the above strategies will be executed to select another index servers.

  The strategy a) ensures that every request between index server and data server will be reached within one hop, which reduces the network delay. Then, by distributing requests into racks, the average workload of index server can be reduced. Finally, strategy b) and c) ensure that the workloads of index

servers keeps low during runtime, so that the possibility of the node crash will be reduced. To evaluate the workload, Eq. 2 is used to get the actual mark for a certain data server:

$$Workload = \alpha * U_{CPU} + \beta * U_{MEM} \quad \text{Eq. 2}$$

Where:

1) Workload: the mark of workload;
2) U_{CPU}: the rate of CPU utilization, represented by % ;
3) U_{MEM}: the rate of Memory utilization, represented by %;
4) α , β : weights for the criteria, both are positive values and $\alpha + \beta = 1$;

The data server with the lowest workload in every rack will be selected as index servers. To reduce the complexity, all index servers will store a full copy of global index, and the master server will push any update to the index to all index servers, so that every replica will be consistent during runtime.

- Caching Strategy

To improve the IO performance, every data server is equipped with both write caches and read caches. The basic data structure of both caches is hash-map with the key of id and value of related object. Since the memory space is limited, it is vital to select the right data to cache. Next the caching strategies will be described respectively.

- Write Cache

The data servers use write cache to reduce the execution delay of writing a vertex. Whenever a vertex needs to be stored on the data server, instead of being written directly onto HDFS, it is put into the write cache. And once the write cache is full, all its content will be flushed onto HDFS. As can be seen from the previous discussion, to improve the performance of HDFS, multiple vertices are grouped into one HDFS file. To this end, the size of write cache is set equal to that of one group. So the vertices in the cache can be stored exactly into one HDFS file, which improves the performance of cache flushing.

- Read Cache:

Like write cache, the read cache is used by data servers to improve the performance of reading vertex descriptions and related datasets. To read the data from a vertex, the data server need to read vertex description, graph schema, data schema, and related dataset in order. To improve the read

performance of above objects, 4 different caches are built to cache the objects accordingly. The respective caching strategies for different kinds of objects are described as follows:

a) Graph schema and data schema: The same space of cache is allocated for both schemas. Whenever a schema is queried, it is inserted into the relative cache for further querying. When either cache is full, the swapping strategy is LRU: the schema that is least recently queried will be swapped-out. Since the neighbors of a certain vertex are likely to be the same type, this strategy reduces the query time for frequent-used schemas.

b) Vertex descriptions and vertex data: Like the caches for graph schema and data schema, the vertex descriptions (see Section III.B) and data are cached for further read. And the time for cache insertion is the same. The difference is the cache swapping strategy. Since the vertices are heterogeneous, for every kind of vertex, the user may have different expectation of query delay. To this end, every kind of vertex (identified by graph schema's id) is marked with a predefined priority. And the vertex with lower priority should be swapped-out first. However, to prevent the vertices with low priority from being swapped-out all the time (the starvation problem), the priority should be adjusted by statistics data of caching hit in the runtime. Under such considerations, a self-adaptive caching strategy is designed. Eq. 3 describes the adjustment of priority:

$$Cache_{priority} = \alpha * (\beta_1 * x_1 + \beta_2 * x_2) + \gamma * y \qquad \text{Eq. 3}$$

Where:

1) Cache_priority: the caching priority for a kind of vertex;
2) x_1 : the times of querying for the kind of vertex;
3) x_2 : the ratio of caching miss for the kind of vertex;
4) β_1, β_2: weights for the runtime statistics data, both are positive values and $\beta_1 + \beta_2 = 1$;
5) α , γ : weights for the criteria, both are positive values and $\alpha + \gamma = 1$;

Eq. 3 balances the user-defined priority and real statistics data of cache in the runtime. And both the vertices with a high user-defined priority or being missed too many times will be kept longer in the cache. For the vertex

descriptions, the item stored in the cache is the whole vertex with all the descriptions, and the graph schema's id is used to determine the cache priority; and for the vertex's attribute data, the item stored in the cache is the data for single attribute of a vertex, and the attribute name is used to determine the cache priority.

4.4. Data system testing and analysis

The data system described above has been tested in a fraud detection application. A heterogeneous knowledge management system has been built in order to analyze the historical data and find potential patterns for fraudulent transactions.

In the prototype of this system, ver 50,000 accounts related to over 2 million transactions are analyzed. The heterogeneous knowledge analysis was then undertaken for three different node types: Create Account, Cash Transaction and Credit Transaction. An index was added for the transaction dataset. In order to see the performance of the whole system, 5 different size datasets: 180M, 1.8G, 5G, 36G and 120G are tested. The environment of the testing system is: 4 nodes cluster system, each node has a 1 core 2.7GHz CPU and 2G memory.

Figure 9 and figure 10 show the time taken to build the index. The x-axis of figure 9 is the size of dataset in MB, the y-axis is the time in seconds. From figure 9, the time of building index would rise along with the size of data set. But the ratio of index construction time and average query time in figure 10, the ratio will be comparatively stable, and never exceed 6. It is quite acceptable as it will not be a big burden for a frequently queried dataset.

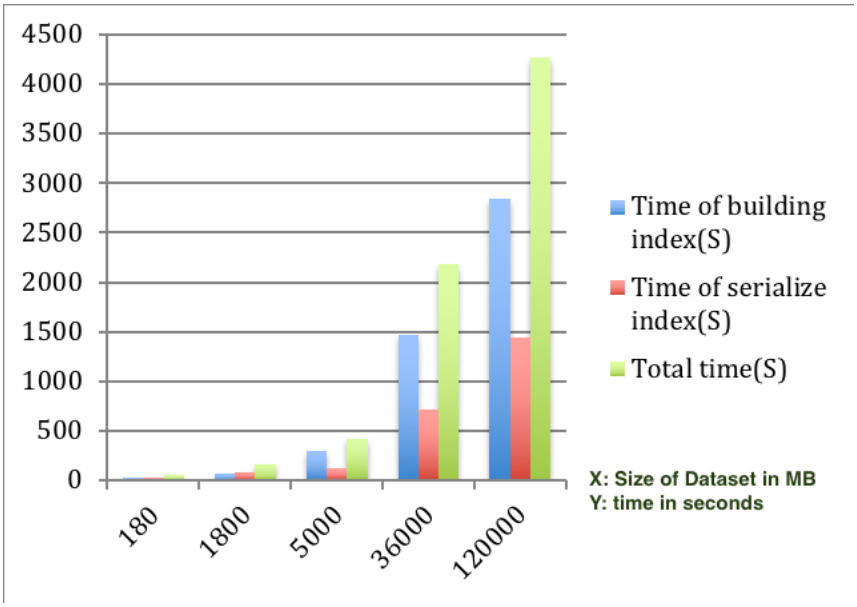


Fig 9. Time of building the index

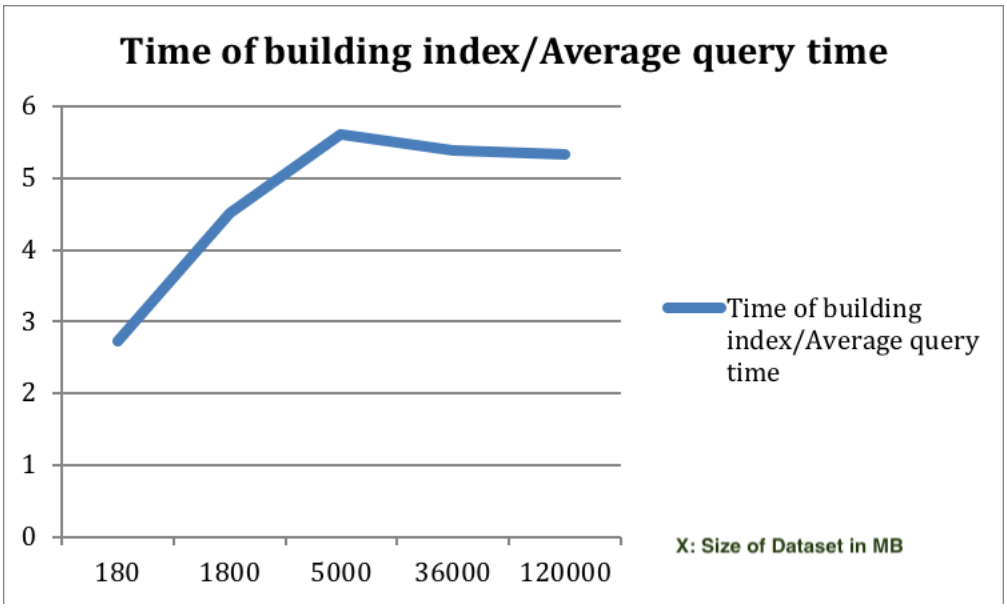


Fig 10. Ratio of building index/average query time

Figure 11, figure 12 and figure 13 show the query time of different size of dataset. The dataset in figure 11 is 5G, 36G in figure 12 and 120G in figure 13. The x-axis of figure 11, 12 and 13 is the ratio of query result size/total dataset size. The y-axis is the query execution time and the total jobs runs on the system.

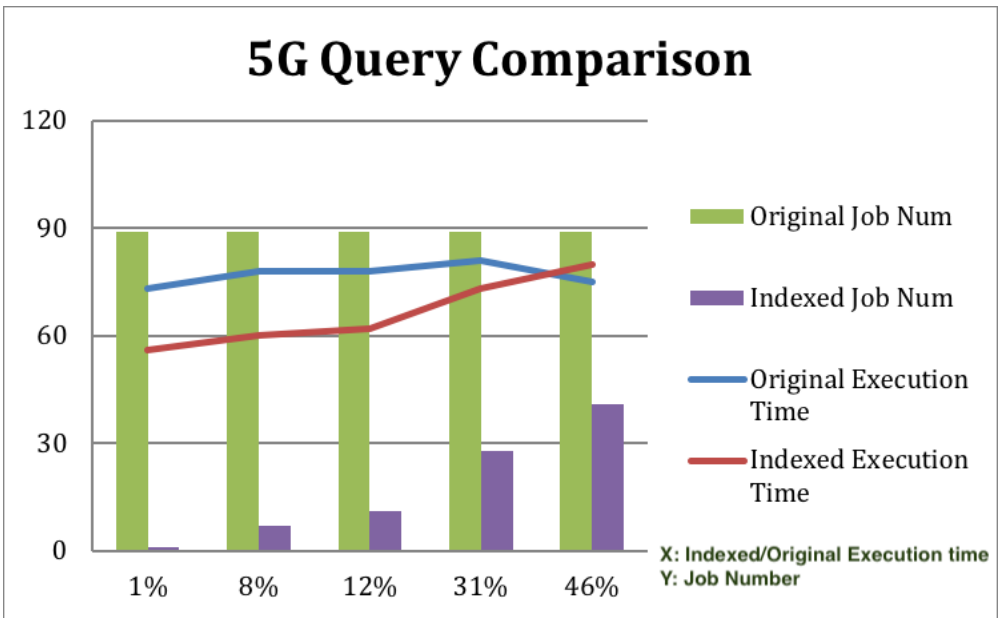


Fig 11. 5G Dataset query comparison

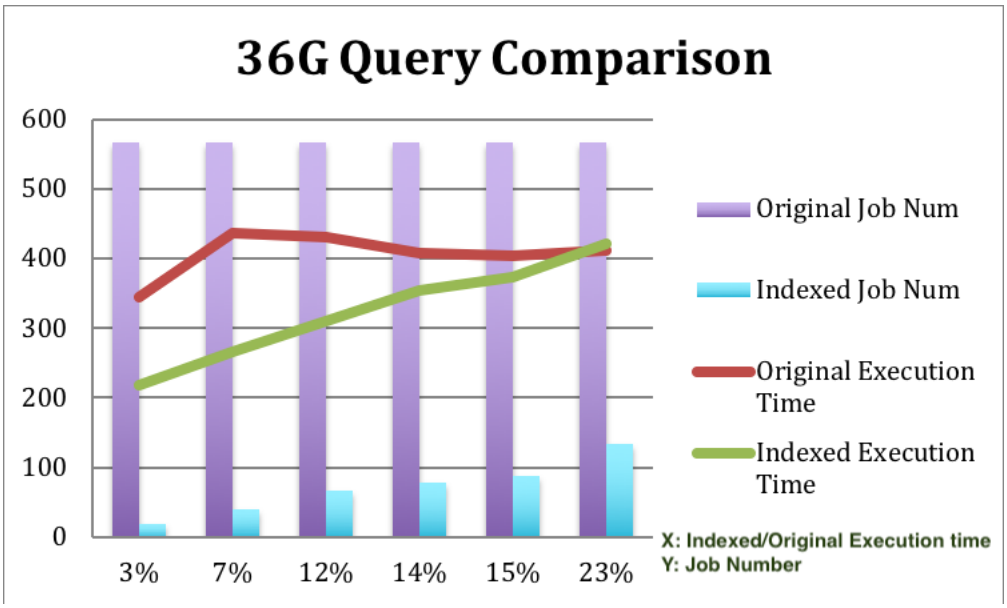


Fig 12. 36G Dataset query comparison

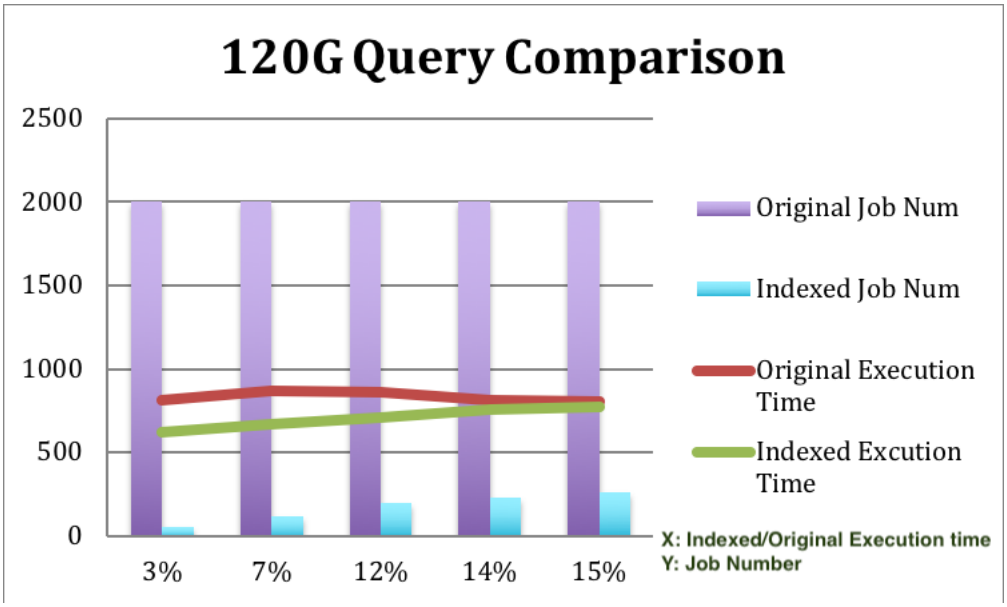


Fig 13. 120G Dataset Query Comparison

From the end-end execution time for parallelized query, in figure 11, because the dataset is not very large, the performance of indexed system is not very different from the original Hadoop system, even when the query result set is large, the total query execution time is a bit longer than the original one, but the total jobs running on the system is significantly decreased. In figure 13 with the rise of dataset volume, the execution time is increased by about 20% percent, but the larger volume of the query result, the less gain you can get from the indexed system. But because with the indexed system, the dataset can be located exactly rather than parse the whole dataset, the total running jobs will decrease dramatically.

5. Analysis and Design of Knowledge Clustering Subsystem

5.1. Knowledge clustering subsystem analysis

A heterogeneous knowledge graph $G = (V, E, A, W)$ is modeled as a directed graph, where each node $v \in V$ in the network corresponds to an knowledge object, and each link $e \in E$ corresponds to a relationship between the linked knowledge objects, with its weight denoted by $W(e)$. Each node v has its attributes $\{a_i \cdots a_j\} \in A$.

Different from the traditional network definition, the objects and links in heterogeneous networks are associated with explicit type information to distinguish the semantic meanings, namely, a mapping function from object to object type (knowledge type), $\tau: V \to K$, and a mapping function from link to link type, $\varphi: E \to R$. K is the object type (knowledge type) set, and R is the link type set, or the relation set, which provides linkage guidance between nodes. Notice that, if a relation exists from type A to type B, denoted as ARB, the inverse relation R^{-1}holds naturally for BR^{-1}A. For most of the times, R and its inverse R^{-1} are not equal, unless the two types are the same and R is symmetric. Attributes are associated with objects, such as the location of a user, the text description of a book, the text information of a blog, and so on. In this setting, attributes across all different types of objects are considered as a collection of attributes for the network, denoted as $A = \{A_1, \cdots A_t\}$, in which we are interested only in a subset for a certain clustering purpose. Each object $v \in V$ contains a subset of the attributes, with observations denoted as $v[A] = \{a_{v1}, a_{v2}, \cdots, a_{vn}, N_{Av}\}$, where N_{Av} is the total number of observations of attribute A attached with object V. Notice that, some attributes can be shared by different types of objects, such as the text and the location attribute; while some other attributes are unique for a certain type of objects, such as the clip time length for a video. V_a is used to denote the object set that contains attribute A.

In an heterogeneous knowledge graph clustering subsystem, there are several challenges under this scenario. First, the massive numbers of attributes may concert to the object, if clustering from a sub set of nodes: $\{v_i, \cdots, v_i\}$, all the attributes $\{a_{vi}, \cdots, a_{vj}\}$ will be counted. If the clustering based on the $\{a_{vi}, \cdots, a_{vj}\}$ dimensional matrix, then we have to face a lot of missing values in the matrix and the dimensions in the matrix which are not purely independent. If separate it into different groups, the relationship and weighting of each group is also hard to set. Second, there are different kinds of linkages in the heterogeneous graph, each linkage representing a particular connection between nodes. But there are so many relationships in the graph it is very difficult to count the importance of the linkage and how to count the importance in numbers is also very hard.

For the first problems, group the $\{a_{vi}, \cdots, a_{vj}\}$ into several groups based on the sematic meaning of the attributes. Inside the group, An entropy based feature selection method is proposed to pick up the key attributes in the group. A Non-Euclidean geometry weighting algorithms is proposed to combine different groups together. For the second problem, an entropy based spreading algorithm will be proposed to spread the linkage of the nodes.

5.2. Heterogeneous knowledge graph clustering subsystem design

5.2.1. Entropy based feature selection and Non-Euclidean geometry weighting algorithms

Given two continuous random variables x and y, their mutual information is defined as below [Cover and Thomas 1991]:

$$I(X;Y) = \int_{\Omega y} \int_{\Omega x} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} dx\, dy \qquad \text{Eq. 4}$$

In equation 4, p(x) and p(y) are marginal probability density functions of X and Y respectively and p(x,y) is the joint probability density function of X and Y. In data management and knowledge sharing system, the relevance of one explanatory attribute with the response attribute/object can be described by the mutual information value of these two attributes and the larger mutual information value represents that these two attributes are more relevant. The

redundancy among explanatory attributes also can be evaluated by mutual information value. However, the larger mutual information value is, the more redundant these explanatory attributes are.

Battiti (Battiti 1994) proposed the mutual information feature selection (MIFS) method to handle feature selection as follows. The basic idea of the MIFS method is to use the relevance value to subtract the redundancy value. Kwak and Choi (Kwak and Choi 2002) proposed MIFS-U method in which he made a better estimation of redundancy value by utilizing the entropy and the mutual information. Peng et al. (Peng, Long et al. 2005) proposed an alternative choice of the parameter β in his mRMR method. He described the redundancy of features by averaging the mutual information of the candidate feature and each selected feature.

The same problem of MIFS and MIFS-U approaches is that the left term and the right term in the equations are not comparable. mRMR method has solved the problem partly by dividing the sum with the cardinality of the set S. Estevez (Estévez, Tesmer et al. 2009) proposed NMIFS method, which is another mutual information feature selection method. He involved the normalized mutual information in the feature selection method to restrict MI's values to the range [0,1]. $NI(f_i; f_\alpha)$is the normalized mutual information of i and feature α .

$$I(C; f_i) - \frac{1}{\|S\|} \sum_{f_\alpha \in S} NI(f_i; f_\alpha) \qquad \text{Eq. 5}$$

$$NI(f_i; f_\alpha) = \frac{I(f_i; f_\alpha)}{\min\{H(f_i), H(f_\alpha)\}} \qquad \text{Eq. 6}$$

The case based reasoning method selects the most nearby cases and use their efforts to estimate the effort of a new software project.

The most widely used measurement is the un-weighted Euclidean distance (UED). It regards the features of a project as the axes and calculates the standard Euclidean distance based on the differences between each corresponding feature. An improved form of the Euclidean distance is the weighted Euclidean distance (WED). It adds additional weights to the computation of the Euclidean distance that assigns different features with different importance according to the specified weights. Mendes et al. (Mendes, Watson et al. 2003) use Pearson correlation coefficient to determine the importance of features. Angelis et al. (Angelis and

Stamelos 2000) proposed the weights are defined as the reciprocal of the standard deviation of the features or the reciprocal of the range of the features. Auer et al. (Auer, Trendowicz et al. 2006) use an extensive search algorithm to find the optimal weight values, whereas (Huang and Chiu 2006)and (Li, Xie et al. 2009) adopt genetic algorithms to achieve this. The common idea of these approaches is that they try to use some learning methods to find better weight values for the weighted Euclidean distance.

Another widely used measurement is the Mahalanobis distance, which is proposed in (Mahalanobis 1936). It is a typical non-orthogonal space distance where the matrix Σ−1 is used to characterize the non-orthogonal space. The matrix makes it possible to take the relevance among features into the distance computation. Our proposed NeGW approach shares a similar idea with this. Thus the Mahalanobis distance no longer considers the features to be probabilistic and independent to each other， which is more suitable for real-life software projects data. Such advantages make the Mahalanobis distance useful in many areas including detection of outliers, the selection of calibration samples from a large set of measurements and clustering techniques such as the nearest neighbor method (De Maesschalck, Jouan-Rimbaud et al. 2000).

In data management and knowledge sharing system, normalize mutual information on both terms are needed. An adapted feature selection method has been used NMIFS(INMIFS)(Thang and Lee 2010)archive that.

$$NI(C; f_i) - \frac{1}{\|S\|} \sum_{f_\alpha \in S} NI(f_i; f_\alpha) \qquad \text{Eq. 7}$$

The INMIFS eliminates the unbalance of the relevance and redundancy by applying normalized mutual information on the left and the right term in order to make the calculated value of both redundancy and relevance confined to [0,1].

In the filter stage, the INMIFS method is performed by the following procedure:

(1) Initialization: set F← initial set of n feature; set S ← empty set.

(2) Computation of the MI value between each candidate feature and the response feature: For each fi∈ F , compute $I(C; f_i)$.

(3) Selection of the first feature: Find the feature fit hat maximize the $I(C; f_i)$, set F ← F\$\{f_i\}$ and set S ←$\{f_i\}$.

(4) Greedy Selection: Repeat until |S| = k.

(a) Computation of the MI between variables: For each pair of features $f_i; f_\alpha$ with$f_i \in$ F and $f_\alpha \in$ S, compute $I(f_i; f_\alpha)$ if it is not yet available.

(b) Selection of the next feature: Choose the feature fi that maximizes the value of equation as follows, then set F ← F\$\{f_i\}$ and set S ←$\{f_i\}$.

$$max\left\{NI(C; f_i) - \frac{1}{\|S\|}\sum_{f_\alpha \in S} NI(f_i; f_\alpha)\right\} \quad \text{Eq. 8}$$

(5) Output the set S containing the selected features.

In the wrapper stage, the task is to determine the optimal m number. Suppose there are n candidate features to be selected in the data set, the INMIFS method using incremental selection produces n sequential feature sets Then compare all these n sequential feature sets to find the set that can minimize the MMRE value of the training set. Finally, m is the optimal number of features and the set S_m is the optimal feature set. After obtaining the optimal feature set S_m, weighted Euclidean distance will be applied to calculate the distance between a specific historical case and the new case. If feature i exists in set S_m, the weight of feature is 1, otherwise it is 0.

As discussed before, most attributes in data management and knowledge sharing have certain relevance with each other and usually it is hard to find many attributes that are completely independent of each other. Thus, a Non-Euclidean geometry weighting (NeGW) measurement is proposed for the case based reasoning method to deal with the relevance between attributes are considered in the distance equation. The key ideas in NeGW is that mutual information has been used to define the angles between attribute axis. The mutual information shows the statistic linkage between the attributes. If the two attributes are independent, the mutual information between them should be 0, the two attributes' axis will be orthogonal. The higher the mutual information is, the more dependence they have, the two attributes' axis should be closer and parallel. So, the mutual information can be used as the angles between attribute axes.

The NeGW is defined in Eq. 9, the concepts of relevance and redundancy are used as the basis for defining proper **δ**. The parameter **δ** will contain two parts: the relevance part, which is the impact of the attribute to the estimation of the effort and the redundancy part, which is the relevancy between the attribute

themselves. The attribute that are of high relevance to the effort but of low redundancy with other attribute are considered to be important for the estimation. So, the idea of defining **δ** in NeGW here is to make the computed distances closer for the cases that have more similarity in attribute of high relevance and low redundancy.

$$NeGW(s,p) = \sqrt{(x_s - x_p)^{\mathrm{T}} \delta (x_s - x_p)}$$

$$\delta = \begin{pmatrix} \delta_{11} & \delta_{12} & \dots & \delta_{1n} \\ \delta_{21} & \delta_{22} & \dots & \delta_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{n1} & \delta_{n2} & \dots & \delta_{nn} \end{pmatrix} \quad \text{Eq. 9}$$

Then the non-orthogonal space distance in Eq. 9 can be properly expressed using the above defined relevance part and redundancy part. On the one hand, the relevance of attribute to the effort can be regarded as the attribute weight that shows how important the attributes are for the estimation of the effort. On the other hand, the redundancy among attribute describes the angles between different attribute axes. The attribute relevance will stretch the corresponding axes to makes the ones with higher relevance more distinguishable in distance computation, whereas the redundancy of attribute will change the angles between attribute axes and make independent attribute more important. Thus, the space is transformed into a non-orthogonal space and the distance is extended to the non-orthogonal space distance. Assume that the relevance, namely attribute weights, is donated by ω where ωi is the relevance of attribute i with the effort, and the redundancy, namely the angles between attribute axes, is donated by θ where θij is the angels between attribute i and attribute j. Then the NeGW can be expressed as in Eq. 10 and it can be further transformed to Eq. 11 that is in the same form as the original definition of NeGW in Eq. 9.

$$NeGW(s,p) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} \left(\omega_i * (x_{si} - x_{pi})\right)\left(\omega_j * (x_{sj} - x_{pj})\right) \cos\theta_{ij}}$$

$$= \sqrt{(x_s - x_p)^{\mathrm{T}} \begin{pmatrix} \omega_1 * \omega_1 & \omega_1 * \omega_2 * \cos\theta_{12} & \dots & \omega_1 * \omega_n * \cos\theta_{1n} \\ \omega_2 * \omega_1 * \cos\theta_{21} & \omega_2 * \omega_2 & \dots & \omega_2 * \omega_n * \cos\theta_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_n * \omega_1 * \cos\theta_{n1} & \omega_n * \omega_2 * \cos\theta_{n2} & \dots & \omega_n * \omega_n \end{pmatrix} (x_s - x_p)} \quad \text{Eq. 10}$$

Then we get that $\boldsymbol{\delta_{ij} = \omega_i * \omega_j * \cos\theta_{ij}}$ So it is reasonable that the **δ** in NeGW be defined as a combination of the attributes relevance and redundancy

part. Then a good way to define these two parts and the normalized mutual information is adopted in our approach. In NeGW, the relevance of attributes will be defined by using the normalized mutual information between the attributes and the effort. The redundancy of attributes will be defined by calculating the mutual information among the attributes. Here we donate the mutual information between feature i and feature j as $I(f_i; f_j)$ and the joint entropy of feature i and feature j as $H(f_i; f_j)$. According to how the relevance part and redundancy part are combined together in the equation, it is further classified into four different approaches.

(1). The relevance-based definition. It is defined in Eq. 12 where only the relevance of attributes is considered. (NeGW-Relv)

$$\delta_{ij} = \frac{I(f_i, f_j; C)}{H(f_i, f_j; C)} \quad \text{Eq. 11}$$

(2). The redundancy-based definition. It is defined in Eq. 13, δ_{ij} is defined only by the angles between attribute axis instead of the weights of features. Highly redundant features will have small impact on the distance calculation and vice versa. (NeGW-Redu)

$$\delta_{ij} = 1 - \frac{I(f_i, f_j)}{H(f_i, f_j)} \quad \text{Eq. 12}$$

(3). The redundancy and relevance-based definition. It is a combination of the above two definitions and both the relevance and redundancy part of the features have been considered. Furthermore, two forms of combination are defined as follows. The difference form uses subtraction to connect the relevance part and the redundancy part whereas the quotient form uses a division.

Besides, to keep the values in a proper range and to avoid dividing by zero they are transformed and finally defined as in Eq. 14 and Eq. 15 respectively. (NeGW-RRD)

$$\delta_{ij} = 1 + \frac{I(f_i, f_j; C)}{H(f_i, f_j; C)} - \frac{I(f_i, f_j)}{H(f_i, f_j)} \quad \text{Eq. 13}$$

(4). (NeGW-RRQ)

$$\delta_{ij} = \frac{I(f_i, f_j; C)}{H(f_i, f_j; C)} \left(1 - \frac{I(f_i, f_j)}{H(f_i, f_j)}\right) \quad \text{Eq. 14}$$

The NeGW-RRD uses difference form to combine the relevance part and the

redundancy part whereas the NeGW-RRQ uses quotient form. Besides, to keep the result in a reasonable range and to avoid dividing by zero error they are finally transformed to the following forms in Eq. 14 and Eq. 15. Both of them help increase the impact of highly relevant but lowly redundant attributes in the NeGW distance computation. Thus, the cases that are more similar in such attributes are more likely to have smaller distances and can be more easily retrieved as similar cases in the case based reasoning method.

5.2.2. PSO-based NeGW Optimization

The above discussed NeGW method depends only on mutual information between features to describe projects similarities. But we assume that such definition is insufficient to be optimal due to the real complexity so further adjustments and optimization are necessary to make the distance definition better fit real situation. Thus, this PSO-based NeGW optimization method is proposed and it will be explained in detail in this section. PSO, the abbreviation for particle swarm optimization, will be discussed in the following section. The basic idea of this PSO based method is to use this algorithm to optimize the matrix in NeGW defined in Eq.9 so as to adapt the distance definition to different dataset. The original NeGW method provides the initial matrix for training and optimization. Based on this, PSO is used to train and optimize the matrix with the goal of minimizing the estimation error. The optimization result is a new matrix δ' which is learned from the training data and can better describe similarities. Finally, based on this optimized distance, estimation accuracy can be notably improved.

5.2.2.1. Particle Swarm Optimization (PSO)

Particle swarm optimization (PSO) is an optimization technique proposed by Kennedy (Kennedy 2010) that tries to find the optimal solution by iteratively searching the problem space in certain patterns. It is aimed at producing computational intelligence by exploiting simple analogues of social interactions, rather than purely individual cognitive abilities (Poli, Kennedy et al. 2007). It is initially designed for simulation of social behaviors like bird flocks or fish schools and later be adopted as a useful optimization technique in many fields. PSO can be

easily adopted for continuous space optimization problems as it requires few assumptions of the problem space. Also it can be more computationally efficient than genetic algorithms in some cases according to (Bardsiri, Jawawi et al. 2012).

For a D-dimensional problem space, PSO contains a number of S particles, each of which is a candidate solution for the problem. There is an objective function $g: R^D \rightarrow R$. Then the PSO algorithm is a minimization problem that iteratively moves the particles in the search space to find an optimal solution that can minimize the objective function. At any time t, each particle i has a position $\overrightarrow{x_i}$ and a velocity $\overrightarrow{v_i}$, which are two D-dimensional vectors. $\overrightarrow{x_i}$ is the current candidate solution of this particle and $\overrightarrow{v_i}$ is the current velocity of this particle moving in the search space. A best position $\overrightarrow{p_i}$ of each particle i and a global best known position $\overrightarrow{p_g}$ of all the particles are also stored. The position and velocity of each particle is adjusted according to Eq. 16 in each iteration.

$$\begin{cases} \overrightarrow{v_i}(t+1) = \omega\overrightarrow{v_i}(t) + \overrightarrow{U}(0,\phi_1)(\overrightarrow{p_i} - \overrightarrow{x_i}(t)) + \overrightarrow{U}(0,\phi_2)(\overrightarrow{p_g} - \overrightarrow{x_i}(t)) \\ \overrightarrow{x_i}(t+1) = \overrightarrow{x_i}(t) + \overrightarrow{v_i}(t+1) \end{cases} \quad \text{Eq. 15}$$

In Eq. 16, $\overrightarrow{U}(0,\phi)$ is a vector of D random numbers each of which has a uniform distribution of $\overrightarrow{U}(0,\phi)$. So basically, there are three parameters to be set: ω, ϕ_1 and ϕ_2. ω is known as the inertia weight which determines how much the particles will follow the original path in each iteration. ϕ_1 and ϕ_2 are called acceleration coefficients where ϕ_1 controls the cognitive part of each particle while ϕ_2 determines the influence of social interactions between particles. Large ϕ_1 may cause the algorithm fall into local minimum while large $\phi_2$may make it hard to converge.

5.2.2.2. Optimize NeGW using PSO

To use PSO for NeGW optimization, we adapt this optimization algorithm to our problem. As described above, the basic PSO algorithm is to optimize a **D** dimensional vector but in our NeGW optimization problem, the matrix δ defined in Eq. 9 needs to be optimized. So the basic PSO algorithm needs to be slightly modified to cope with such problem. Our solution is that the upper triangle of the matrix δ is expanded as a vector (since δ is a symmetric, it is the same to choose either upper or lower triangle of the matrix) and this vector is served as the input

of the PSO algorithm. Then after the optimization process, an optimized vector is produced and convert back to a matrix which will be regarded as the optimized matrix δ'. The mapping between the matrix δ and the input/output vector can be illustrated in Eq. 17.

$$\delta: \begin{pmatrix} \delta_{11} & \delta_{12} & \dots & \delta_{1n} \\ \delta_{21} & \delta_{22} & \dots & \delta_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{n1} & \delta_{n2} & \dots & \delta_{nn} \end{pmatrix} \xleftrightarrow{convert} \vec{v}: \begin{pmatrix} (\delta_{11} & \delta_{12} & \dots & \delta_{1n} \\ & \delta_{22} & \dots & \delta_{2n} \\ & & \dots & \dots \\ & & & \delta_{nn}) \end{pmatrix} \quad \text{Eq. 16}$$

For our problem, δ is a $\boldsymbol{n} \times \boldsymbol{n}$ matrix so the corresponding to-be-optimized vector is of length $\boldsymbol{D' = n(n+2)/2}$. Then the problem is converted into a PSO problem that searches best solution in a $\boldsymbol{D'}$-dimensional space. In the context of our problem, the problem space of this optimization method is composed of all the possible $\boldsymbol{n} \times \boldsymbol{n}$ matrices that can be used in NeGW definition and the goal is to find an optimal one. The current position of each particle in PSO can be regarded as a possible solution it currently finds. According to previous texts, the local and global so far best solutions are stored and PSO tries to find the optimal solution by iteratively moving particles in the search space, which is in effect trying different possible solutions.

More specifically, when the matrix δ is first computed according to the NeGW definition in Eq. 9. Then δ is converted to a vector $\vec{v}$ and $\vec{v}$ is served as the input for the particle swarm optimizer which then outputs the optimized vector $\overrightarrow{v'}$. $\overrightarrow{v'}$ is finally converted back to a matrix δ' and it is just the optimized matrix we are going to use for defining the new distance measure. The new distance formula is shown in Eq. 18 where δ' is the optimized matrix instead of original matrix defined in NeGW.

$$NeGW(s,p) = \sqrt{(x_s - x_p)^{\mathrm{T}} \delta' (x_s - x_p) 0} \quad \text{Eq. 17}$$

Since the goal of our problem is to increase estimation accuracy, we define the objective function $g: R^D \to R$ of the PSO algorithm to minimize the average relative estimation error which is measured based on the evaluation metric MMRE.

The pseudo code for this optimization process is illustrated in Algorithm 1. In

the pseudo code, procedure **Optimize()** shows the overall optimization process and procedure **ObjectiveFunction()** is the objective function for the optimizer. Algorithm 2 gives an outline of the PSO algorithm, which is invoked in the optimization process.

Algorithm 1 Pseudo code for the optimization process

procedure OPTIMIZE( ) ▷ Optimization process
  for $i \leftarrow 1$ to n **do** ▷ Initialize according to (8)
    for $j \leftarrow 1$ to n **do**
      $\delta_{ij} = 1 + \frac{I(f_i,f_j;C)}{H(f_i,f_j,C)} - \frac{I(f_i;f_j)}{H(f_i,f_j)}$
    end for
  end for
  $\delta' \leftarrow \delta$
  $\overrightarrow{v} \xleftarrow{convert} \delta'$
  $\overrightarrow{v}' \leftarrow$ **PSO**$(\overrightarrow{v})$ ▷ PSO-based optimization
  $\delta' \xleftarrow{convert} \overrightarrow{v}'$
return δ' ▷ The optimized matrix
end procedure

procedure OBJECTIVEFUNCTION(δ') ▷ The objective function
  $D(s,p) \leftarrow \sqrt{(\vec{x_s} - \vec{x_p})^{\mathrm{T}} \delta' (\vec{x_s} - \vec{x_p})}$
  for each project i in training set **do**
    estimate cost $\hat{y}_i$ using $D(s,p)$
    $MRE_i \leftarrow |\hat{y}_i - y_i|/y_i$ ▷ y_i is the real cost
  end for
  $MMRE = \frac{1}{n} \times \sum_{i=1}^{n} MRE_i$
return $MMRE$
end procedure

Algorithm 2 Pseudo code for the PSO algorithm

procedure PSO($\overrightarrow{x_{inttial}}$) ▷ PSO Algorithm
  $g \leftarrow$ **ObjectiveFunction**
  for each particle $i \leftarrow 1$ to S **do** ▷ Initialization
    initialize velocity: $\overrightarrow{v_i}$
    initialize position: $\overrightarrow{x} \leftarrow \overrightarrow{x_{inttial}}$
  end for
  repeat
    for each particle $i \leftarrow 1$ to S **do** ▷ Initialization
      initialize u_1: let u_1 $\overrightarrow{U}(0, \phi_1)$
      initialize u_2: let u_2 $\overrightarrow{U}(0, \phi_2)$
      $\overrightarrow{v_i} \leftarrow \omega \cdot \overrightarrow{v_i} + u_1 \cdot (\overrightarrow{p_i} - \overrightarrow{x_i}) + u_2 \cdot (\overrightarrow{p_g} - \overrightarrow{x_i})$
      $\overrightarrow{x_i} \leftarrow \overrightarrow{x_i} + \overrightarrow{v_i}$
      if $g(x_i) < g(p_i)$ **then**
        $p_i \leftarrow x_i$
        if $g(p_i) < g(p_g)$ **then**
          $p_g \leftarrow p_i$
        end if
      end if
    end for
  until termination criteria is met
return p_g ▷ The optimized vector
end procedure

Fig 14. The pseudo code for optimization process and outline of PSO algorithm

Besides, PSO parameters are set according to the suggestions of the standard particle swarm optimization (SPSO) (Clerc 2010). Since we are not going to have a comprehensive analysis of the effects of different PSO variants and parameters on the cost estimation, this research uses a set of predefined parameter values in all the experiments: $\omega = 0.721$ and $\phi_1 = \phi_2 = 1.193$.

5.2.3. Knowledge graph structure consistency model

From the view of links, the more similar the two objects are in terms of cluster membership, the more likely they are connected by a link. In order to quantitatively measure the consistency of a clustering result with the network structure, a novel probability density function has been used for observing clustering result.

Assume that linked objects are more likely to be in the same cluster, if the link type is of importance in determining the clustering process. As defined in Chapter 3.2.1, there are several relations of the relationship between the two knowledge types: ISA-relation, HSA-relation and SKO-relation.

For the SKO-relation, the two linked objects $\boldsymbol{v_i}$ and $\boldsymbol{v_j}$, their membership probability vectors $\boldsymbol{\theta_i}$ and $\boldsymbol{\theta_j}$ should be similar. Within the same type of links, the higher link weight $W(e)$, the more similar $\boldsymbol{\theta_i}$ and $\boldsymbol{\theta_j}$ should be. Further, a certain link type may be of greater importance, and will influence the similarity to a greater extent.

The consistency with the network G, is evaluated with the use of a composite analysis with respect to all the links in the network in the form of a probability density value. The consistency of the two objects will be spread between them. In the following, we first introduce how the consistency of two cluster membership vectors is defined with respect to a single link, and then how this analysis can be applied over all links in order to create a probability density value.

For a link $\boldsymbol{e} = \{\boldsymbol{v_i}, \boldsymbol{v_j}\} \in \boldsymbol{E}$, with type $r = \boldsymbol{\phi}(\boldsymbol{e}) \in \boldsymbol{R}$ we denote the importance of the link type to the clustering process by a real number $\gamma(r)$. This is different from the weight of the link $W(e)$, which is specified in the network as input, whereas the value of $\gamma(r)$ is defined on link types and needs to be learned. We denote the consistency function of two cluster membership vectors $\boldsymbol{\theta_i}$ and $\boldsymbol{\theta_j}$, with link $\boldsymbol{e}$ under strength weights for each link type γ by a feature function $\boldsymbol{f}(\boldsymbol{\theta_i}, \boldsymbol{\theta_j}, \boldsymbol{e}, \boldsymbol{\gamma})$. Higher values of this function imply greater consistency with the clustering results. In the following, we list several desiderata for a good feature function:

1. The value of the feature function $\boldsymbol{f}$ should increase with greater similarity of $\boldsymbol{\theta_i}$ and $\boldsymbol{\theta_j}$.

2. The value of the feature function $\boldsymbol{f}$ should decrease with greater importance of the link $\boldsymbol{e}$, either in terms of its specified weight $W(e)$, or learned importance $\gamma(r)$. In other words, for the larger strength of a particular link type, two linked nodes are required to be more similar to claim the same level of consistency.

3. The feature function should not be symmetric between its first two arguments $\boldsymbol{\theta_i}$ and $\boldsymbol{\theta_j}$, because the impact from node $\boldsymbol{v_i}$ to node $\boldsymbol{v_j}$ could be different from that of $\boldsymbol{v_j}$ to $\boldsymbol{v_i}$.

The last criterion requires some further explanation. For example, in a citation network, a paper i may cite paper j, because i feels that j is relevant to itself, while the reverse may not be necessarily true. In the experimental section, asymmetric

feature functions will be show to produce higher accuracy in link prediction.

Then a cross entropy-based feature function is proposed, which satisfies all of the desiderata listed above. For a link $\boldsymbol{e} = \{\boldsymbol{v_i}, \boldsymbol{v_j}\} \in \boldsymbol{E}$, with relation type $r = \boldsymbol{\phi}(\boldsymbol{e}) \in \boldsymbol{R}$ the feature function $\boldsymbol{f}(\boldsymbol{\theta_i}, \boldsymbol{\theta_j}, \boldsymbol{e}, \boldsymbol{\gamma})$ is defined as:

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{\theta_i}, \boldsymbol{\theta_j}, \boldsymbol{e}, \boldsymbol{\gamma}) &= -\gamma(r)W(e)\, H(\boldsymbol{\theta_j}, \boldsymbol{\theta_i}) \\ &= \gamma(r)W(e) \sum_{k=1}^{K} \theta_{j,k} \log \theta_{i,k} \end{aligned} \qquad \text{Eq. 18}$$

where $H(\boldsymbol{\theta_j}, \boldsymbol{\theta_i}) = \sum_{k=1}^{K} \theta_{j,k} \log \theta_{i,k}$ is the cross entropy from $\boldsymbol{\theta_j}$ to $\boldsymbol{\theta_i}$, which evaluates the deviation of $\boldsymbol{v_j}$ from $\boldsymbol{v_i}$, in terms of the average coding bits needed if using coding schema based on the distribution of $\boldsymbol{\theta} i$. For a fixed value of $\gamma(r)$, the value of $H(\boldsymbol{\theta_j}, \boldsymbol{\theta_i})$ is minimal and $\boldsymbol{f}$ is maximal, when the two vectors are identical. It is also evident from Eq. 19 that the value of $\boldsymbol{f}$ decreases with increasing learned link type strength $\gamma(r)$ or input link weight $W(e)$. We require $\gamma \geq 0$, in the sense that we do not consider links that connect dissimilar objects. The value of $\boldsymbol{f}$ so defined is a non-positive function, with larger value indicating a higher consistency of the link.

Other distance functions such as KL-divergence could replace the cross entropy in the feature function. However, as cross entropy favors distributions that concentrate on one cluster ($H(\boldsymbol{\theta_j}, \boldsymbol{\theta_i})$ achieves the lowest distance, when $\boldsymbol{\theta_j} = \boldsymbol{\theta_i}$ and $\boldsymbol{\theta_{i,k}} = 1$ for some cluster k), which agrees with our clustering purpose, we pick it over KL-divergence.

Then a log-linear model is proposed to model the probability of Θ given the link type weights γ, where the probability of one configuration Θ is defined as the exponential of the summation of feature functions of all the links in G:

$$p(\Theta|\mathrm{G}, \gamma) = \frac{1}{Z(\gamma)} exp\left\{ \sum_{e=\langle v_i, v_j \rangle \in E} f(\theta_i, \theta_j, e, \gamma) \right\} \qquad \text{Eq. 19}$$

Where γ is the strength weight vector for all link types, $f(\theta_i, \theta_j, e, \gamma)$ is the feature function defined on links of different types, and $Z(\gamma)$ is the partition function that makes the distribution function sum up to 1: $Z(\gamma) = \int_{\Theta} exp\left\{\sum_{e=\langle v_i, v_j \rangle \in E} f(\theta_i, \theta_j, e, \gamma)\right\} d\Theta$. The partition function $Z(\gamma)$ is an integral over the space of all the configurations Θ, and it is a function of γ.

5.3. Experiments and discussions

To validate the performance of different feature selection methods and distance measuring methods, several experiments are designed: the first one is to compare different feature selection methods to evaluate the performance of entropy based feature selection method. The second one is to compare the Non-Euclidean geometry weighting algorithms and Euclidean geometry weighting algorithms. The third one to evaluate the PSO optimized NeGW method and its configuration.

5.3.1 experiments on feature selection method

The purpose of this experiment is to compare different kinds of feature selection methods in the similarity measurement between attributes. The experiments will evaluate the mutual information method in feature selection in software cost estimation.

The similarity measure methods to compare in the experiments is showed in Table 1, there are some statistical methods, correlation coefficient methods and as well as mutual information methods proposed.

Table 1. Similarity Measurement Methods in Attribute Selection

Type	Method
Statistical method	Chi-squared
Correlation method	Pearson Correlation
	Spearman Correlation
Mutual information method	Information Gain
	Information Gain Ratio
	Symmetrical Uncertainty

The experiments parameters setting will be listed in Table 2. The methods isi tested in Desharnais (Sayyad Shirabad 2005) and ISBSG R8 datasets(Group 2003). Different subsets from all the attributions in the dataset is selected to calculate the accuracy of cost estimation. The NeGW method in the case retrieval process is used, and select three recent cased as the most similar historical projects to count the means of the project cost. The entire experiment will use a three-way data split cross validation method.

Table 2. Parameter Setting of software cost estimation experiments

Parameters	DataSet	
	ISBSG R8	Desharnais
All attributes of the dataset	Summarized Work Effort, Function Points, Project Elapsed Time, Organization Type, Business Area Type, Application Type Development Platform, Primary Programming Language, How Methodology Acquired, Recoding Method, Counting Technique, Development Type, Resource Level,	Effort, PointsAjust, Length, Language, ManagerExp, TeamExp
Attribute Numbers	13	6
Project Numbers	306	77
Cross-validation	Three-Way Data Split	
Neighbor Numbers	$K = 3$	
Distance Measurement	NeGW: $D(s,p) = \sqrt{(\overrightarrow{x_s} - \overrightarrow{x_p})^T \delta (\overrightarrow{x_s} - \overrightarrow{x_p})}$	
Cost estimation model	Mean	

Table 3 show the result of MMRE and PRED(0.25) for different similarity measurement methods in attribute selection.

Table 3. Feature Selection in Desharnais

Methods	Attribute Number	$MMRE$	$MdMRE$	$PRED(0.25)$
Chi-Square	1	0.738	0.386822	0.329004
	2	0.690949	0.381187	0.354257
	3	0.717658	0.421512	0.309524

	4	0.83139	0.44601	0.2886
Pearson Correlation	1	0.716013	0.378749	0.339105
	2	0.711198	0.379215	0.336941
	3	0.794554	0.412878	0.316017
	4	0.674231	0.419274	0.306638
Spearman Correlation	1	0.714146	0.367122	0.348485
	2	0.704549	0.385199	0.344156
	3	0.616067	0.370668	0.342713
	4	0.692901	0.428565	0.313131
Information Gain	1	0.716769	0.366299	0.34632
	2	0.715268	0.398721	0.343074
	3	0.740159	0.407752	0.313853
	4	0.782821	0.444439	0.297619
Information Gain Ratio	1	0.71019	0.376839	0.340909
	2	0.720193	0.383489	0.337662
	3	0.711018	0.400684	0.314935
	4	0.824885	0.443592	0.307359
Symmetrical Uncertainty	1	0.732454	0.391924	0.331169
	2	0.695359	0.385255	0.367965
	3	0.725593	0.40766	0.312771
	4	0.811052	0.438519	0.286797

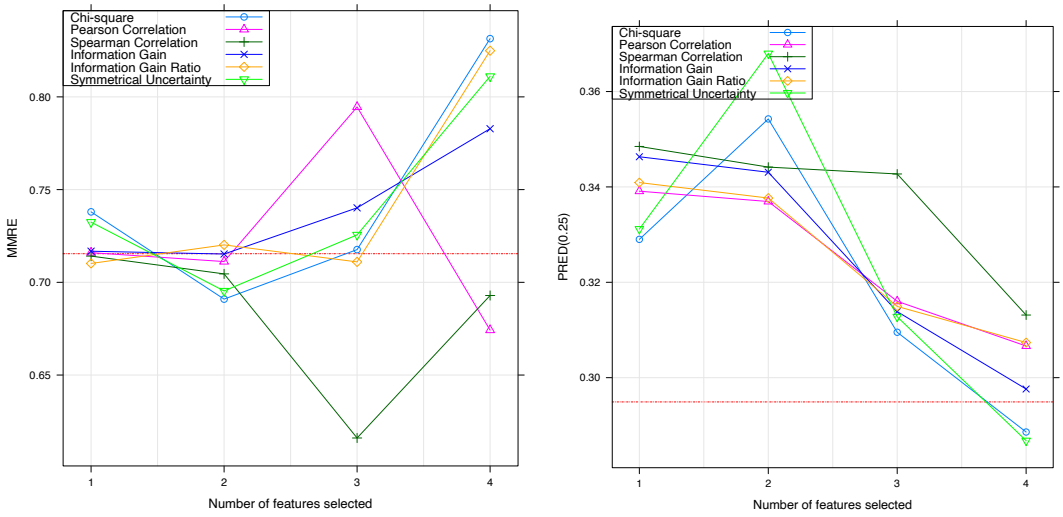


Fig 15. MMRE and PRED(0.25) in Desharnais Dataset

Fig 14 shows the software cost estimation of different similarity measure methods in attribute selection when selecting a different number of attributes. The horizontal dashed line in the figure indicates the estimation accuracy of all attributes in selected case, which can be used as a baseline comparison of various methods. You can see the results of most of the methods relatively similar. For the PRED (0.25), various methods showed similar trends. The chi-square detection (Chi-square Test) and symmetric uncertainty (Symmetrical Uncertainty) method get maximum PRED (0.25) when they select two attributes. For the other methods, they will get maximum PRED (0.25) when they select one attribute. Spearman correlation coefficient method is relatively stable. The PRED (0.25) will have only a little change when selecting 1, 2 or 3 attributes. But when the selected four attributes, the accuracy will decrease rapidly. However, several other methods in the selection of three or four attributes, accuracy are has significant reduction. Overall speaking, after attribute selection, PRED (0.25) has increased, only some methods when we select four attributes, its accuracy will be slightly lower than the reference method without attribute selection. For MMRE, the result is more complicated. When you select one or two attributes, estimation errors of all methods are not very different, but when you select three attributes, Spearman correlation coefficients have the best effect, and when selected four attributes, Pearson correlation coefficients have the best effect. The MMRE error will on an upward trend after selecting more than three attributes for Chi-square testing and cross-correlation methods. Therefore, on the Desharnais data set, Spearman correlation coefficients generally achieved good results, and some of the mutual information methods

(information gain, information gain ratio and symmetrical uncertainty) have very good estimation accuracy when only select a few attributes. Also, due to less number of attributes the dataset, the experimental results it is difficult to fully reflect all the characteristics of each similarity measure method.

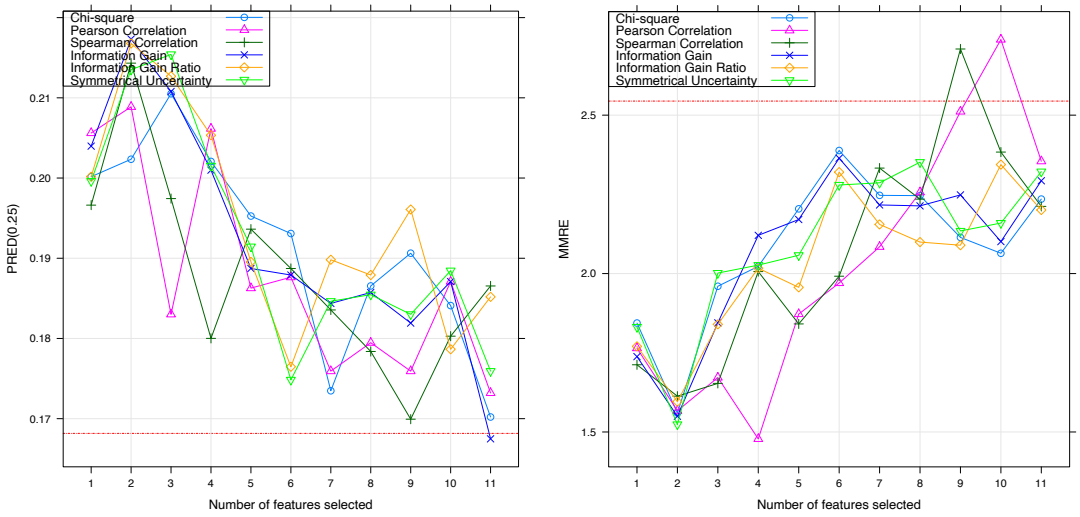


Fig 16. MMRE and PRED(0.25) in ISBSG R8 Dataset

Figure 15 shows the experimental results of ISBSG R8 datasets. Because the number of attributes is more than that in Desharnais, the result shows stronger regularity. For MMRE, the overall estimation error increases substantially with the increase of number of selected attributes. Pearson Correlation reach the minimum estimation error in the choice of three attributes, while other methods to achieve the best results in the selection of two attributes. With the increase of the number of attributes, the estimation errors keep rising. For the mutual information method, the worst estimation error comes with six attributes; for the correlation coefficients method, the worst estimation error comes with 9 or 10 attributes. General speaking, after attribute selection, cost estimation errors are smaller than the methods without feature selection. For PRED (0.25), because it represent the accuracy of the prediction, the trends of PRED (0.25) and MMRE basic contrast with each other, the accuracy of different methods are relatively very close. Most of the methods will reach maximum estimation accuracy with two or three attributes selected. However, the correlation coefficient based methods will decrease a lot for the accuracy with three or four attributes. Similarly, by attribute selection, PRED (0.25) has greatly improved over baseline without attribute selection method. From MMRE and PRED (0.25) Comprehensive view, usually select a smaller number of attributes can get better cost estimation accuracy, in addition, a

method based on the correlation coefficient will have greater ups and downs in the ISBSG R8 data sets, and methods based on mutual information is relatively stable.

5.3.2 experiments on Non-Euclidean geometry weighting algorithms

The purpose of this experiment is to evaluation the Non-Euclidean geometry weighting algorithms in similarity measurement. In this experiments, NeGW-Relv, NeGW-Redu, NeGW-RRD, NeGW-RRQ and standard Euclidean distance are compared.

Table 4. The method description of similarity measurement

Method	Equation	Comment
Euc	$D(s,p)=\sqrt{\sum_{i-1}^{n}\left(x_{si}-x_{pi}\right)^2}$	Standard Euclidean distance
NeGW	$D(s,p)=\sqrt{\left(\overrightarrow{x_s}-\overrightarrow{x_p}\right)^T\delta\left(\overrightarrow{x_s}-\overrightarrow{x_p}\right)}$	Non-Euclidean distance
	NeGW-Relv $\delta_{ij}=\frac{I(f_i,f_j;C)}{H(f_i,f_j,C)}$	Relative based NeGW
	NeGW -Redu $\delta_{ij}=1-\frac{I(f_i;f_j)}{H(f_i,f_j)}$	Redundancy based NeGW
	NeGW -RRD $\delta_{ij}=1+\frac{I(f_i,f_j;C)}{H(f_i,f_j,C)}-\frac{I(f_i;f_j)}{H(f_i,f_j)}$	Diff form of Relative and Redundancy based NeGW
	NeGW -RRQ $\delta_{ij}=\frac{I(f_i,f_j;C)}{H(f_i,f_j,C)}\left(1-\frac{I(f_i;f_j)}{H(f_i,f_j)}\right)$	Quotient from of Relative and Redundancy based NeGW

Table 4 listed all the five methods compared in this experiment. The experiments parameters setting will be listed in Table 5. The methods is also tested in Desharnais(Sayyad Shirabad 2005) and ISBSG R8 datasets(Group 2003). Three recent cased is selectd as the most similar historical projects to count the means of the project cost. The entire experiment will use cross validation method.

Table 5. Parameter Setting of NeGW experiments

Parameters	DataSet	
	Desharnais	ISBSG R8

All attributes of the dataset	Effort, PointsAjust, Length, Langage	Summarized Work Effort, Function Points, Project Elapsed Time, Organization Type, Development Platform, Recoding Method, Counting Technique
Attribute Numbers	4	7
Project Numbers	77	306
Cross-validation	LOOCV	
Neighbor Numbers	$K = 3$	
Cost estimation model	Mean	

Table 6 show the result of MMRE and PRED (0.25) for different weighting measurement.

Table 6. Weighting method in ISBSG R8 and Desharnais

Method	**ISBSG R8**			**Desharnais**		
	$MMRE$	$MdMRE$	$PRED(0.25)$	$MMRE$	$MdMRE$	$PRED(0.25)$
Euc	1.37	0.55	0.24	0.40	0.31	0.39
NeGW-Relv	1.27	0.54	0.24	0.40	0.29	0.45
NeGW - Redu	1.19	0.60	0.22	0.49	0.36	0.40
NeGW -RRD	1.29	0.54	0.25	0.37	0.30	0.47
NeGW -RRQ	1.25	0.51	0.27	0.53	0.36	0.39

First, we look at results ISBSG R8 datasets. Overall, the Non-Euclidean geometry weighting (NeGW) In most cases achieved good results on MRE, MdMRE and PRED (0.25). Specifically, the correlation-based Non-Euclidean geometry weighting has similar result with Euclidean distance (Euc) in PRED (0.25), both of them are 0.24, and for MRE and MdMRE, the results of both are very similar. Also you can see the NeGW-RRD and NeGW-RRQ methods achieve very good result, NeGW-RRD get 4.2% increase in PRED (0.25) than the Euc method, and there is a certain reduction in the MMRE and MdMRE. The NeGW-RRQ achieve 0.27 in PRED (0.25), and also achieved a minimum MdMRE 0.51. On the other hand, NeGW-Redu method is slightly

worse than the Euc method in PRED (0.25) and MdMRE, but it made the best MMRE accuracy.

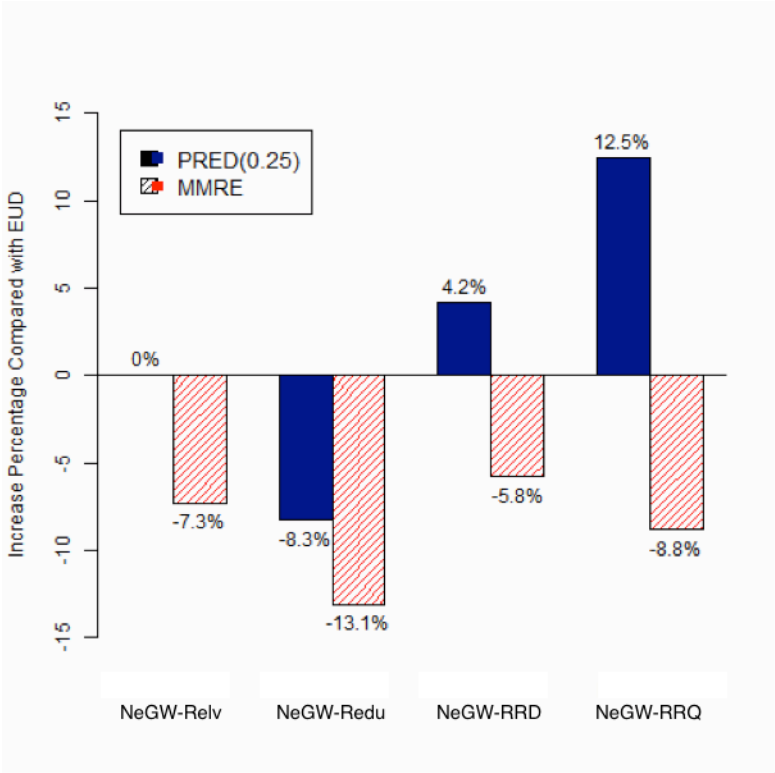


Fig 17. NeGW methods in ISBSG R8 Dataset

For the experimental results Desharnais dataset, the overall performance is similar to the ISBSG R8 dataset. Most of the evaluation on Non-Euclidean geometry weighting (NeGW) methods is better than the Euclidean distance (Euc). Similar, NeGW-Redu is better than the Euclidean distance in PRED (0.25), but reaches better MMRE and MdMRE. For NeGW-Relv method, PRED (0.25) is 15% increase, and a certain MMRE and MdMRE reduced. In addition, NeGW-RRD in this test method to obtain the best prediction accuracy - made the biggest PRED (0.25) values of 0.47, 0.37, and the minimum value of MMRE. But NeGW-RRQ get the same PRED (0.25) with Euclidean distance and the larger MRE and MdMRE.

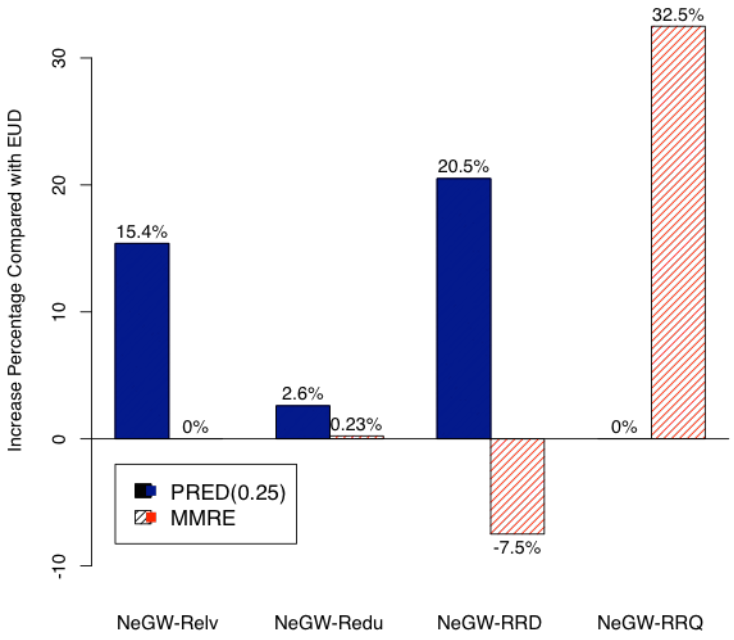


Fig 18. NeGW methods in Desharnais Dataset

From the result shows in the ISBSG R8 and Desharnais datasets, since the increase of PRED (0.25) and the decrease of MMRE means the improvement of the accuracy, NeGW methods in most cases reaches higher accuracy compare with Euclidean distance. From the results of the two datasets, the estimation accuracy on ISBSG R8 dataset is significantly worse than that on Desharnais, the reason is that the ISBSG R8 dataset are from different industries, it contains a collection of different projects in different application areas, the complexity of the projects are very different with each other. In addition, for PRED (0.25) in the Desharnais dataset, NeGW methods perform better than the Euclidean distance, but in the ISBSG R8, NeGW-Redu result worse than the Euclidean distance. For MMRE, on ISBSG R8 dataset NeGW made great accuracy improved, but only on Desharnais dataset NeGW-RRD achieved good results. In addition, in most cases, NeGW-RR (including NeGW-RRD and NeGW-RRQ) method on two data sets is able to achieve better results. Which NeGW-RRD method is relatively more stable, but NeGW-RRQ methods are often able to obtain the best prediction accuracy.

5.3.3 experiments on PSO optimized NeGW algorithms

The purpose of this experiment is to evaluate the PSO optimized NeGW algorithm, PsoNeGW method is tested in cost estimation, compare the different similarity measurement method and distance weighting method in PsoNeGW.

Table 7. Similarity Measurement Method in PSO experiments

Method	Equation	Comment
Euc	$D(s,p) = \sqrt{\sum_{i-1}^{n}(x_{si} - x_{pi})^2}$	Euclidean Distance
Man	$D(s,p) = \sum_{i-1}^{n}(x_{si} - x_{pi})$	Manhattan Distance
Min	$D(s,p) = \left(\sum_{i-1}^{n}(x_{si} - x_{pi})^l\right)^{1/l}$	Minkowski Distance
Mah	$D(s,p) = \sqrt{(\overrightarrow{x_s} - \overrightarrow{x_p})^T \Sigma^{-1}(\overrightarrow{x_s} - \overrightarrow{x_p})}$	Mahalanobis Distance
NeGW	$D(s,p) = \sqrt{(\overrightarrow{x_s} - \overrightarrow{x_p})^T \delta (\overrightarrow{x_s} - \overrightarrow{x_p})}$	Non-Orthogonal Space Distance
	NeGW-RRD $\delta_{ij} = 1 + \frac{I(f_i, f_j; C)}{H(f_i, f_j, C)} - \frac{I(f_i; f_j)}{H(f_i, f_j)}$	Diff form of Relative and Redundancy based NeGW
	NeGW-RRQ $\delta_{ij} = \frac{I(f_i, f_j; C)}{H(f_i, f_j, C)}\left(1 - \frac{I(f_i; f_j)}{H(f_i, f_j)}\right)$	Quotient from of Relative and Redundancy based NeGW
PsoWE	$D(s,p) = \sqrt{\sum_{i-1}^{n}\omega_i(x_{si} - x_{pi})^2}$	PSO-Optimized Weighted Euclidean Distance
PsoNeGW	$D(s,p) = \sqrt{(\overrightarrow{x_s} - \overrightarrow{x_p})^T \delta' (\overrightarrow{x_s} - \overrightarrow{x_p})}$	PSO-Optimized NeGW

Between all the NeGW methods, the NeGW-RRD and NeGW-RRQ method is selected in this experiment because they can get better performance from the preview experiments. For the PsoWE and PsoNeGW, the suggestion from SPSO (Standard Particle Swarm Optimization) (Clerc 2010) is adapted, all the parameters will be listed in Table 8. The optimization target function of PSO is: $g: \delta \rightarrow MMRE$.

Table 8. Parameter Setting of PSO algorithms

Parameter	ω	ϕ_1	ϕ_2
Value	0.721	1.193	1.193

the Desharnais and ISBSG R8 dataset are also used in this experiment. All the parameter of the experiments is listed in Table 9. The entire experiment will use a three-way data split cross validation method.

Table 9. Parameter Setting of SPO experiment

Parameters	DataSet	
	Desharnais	ISBSG R8
All attributes of the dataset	Effort, PointsAjust, Length, Language	Summarized Work Effort, Function Points, Project Elapsed Time, Organization Type, Development Platform, Recoding Method, Counting Technique
Attribute Numbers	4	7
Project Numbers	77	306
Cross-validation	Three-Way Data Split	
Neighbor Numbers	$K = 1,2,3,4,5$	
Cost estimation model	Mean	$y_t = \frac{1}{K}\sum_{i=1}^{K} y_{n_i}$
	Median）	$y_t = median(y_{n_1}, y_{n_2}, \cdots, y_{n_K})$
	IWDM	$y_t = \sum_{i=1}^{K}\left(\frac{1/D(t,n_i)}{\sum_{j=1}^{K}(1/D(t,n_j))} \times y_{n_i}\right)$

We will summarize the result in Table 10. General speaking, the best result can be found when K = 2,3,4 in most cases, However, different methods and for different data sets, each method will has a different trends, there is no a particular K value that we can find best results in all the experimental configuration, it is very difficult to judge what is the best value for the neighbor numbers K. However, for K = 1,5, mostly leads to poor results. One more thing that need to mention is that because the cost estimation depends a lot on history records, the estimation accuracy become unstable since there are too less or too much of the history records.

Table 10. Summarize of the PSO experiments for distance measurement

Methods	Neighbor	Desharnais	ISBSG R8

	Numbers	$MMRE$	$MdMRE$	$PRED(0.25)$	$MMRE$	$MdMRE$	$PRED(0.25)$
Euc	1	0.547	0.378	0.325	2.01	0.663	0.186
	2	0.54	0.361	0.342	2.132	0.666	0.19
	3	0.578	0.365	0.347	2.005	0.652	0.195
	4	0.596	0.382	0.345	2.087	0.645	0.201
	5	0.64	0.399	0.328	2.166	0.659	0.204
Man	1	0.542	0.367	0.338	2.395	0.692	0.182
	2	0.573	0.36	0.35	2.001	0.644	0.198
	3	0.532	0.373	0.348	2.206	0.657	0.201
	4	0.542	0.361	0.359	2.073	0.652	0.199
	5	0.671	0.389	0.341	1.938	0.652	0.192
Min	1	0.56	0.37	0.339	2.859	0.746	0.162
	2	0.607	0.37	0.332	3.834	0.758	0.156
	3	0.64	0.378	0.338	2.511	0.722	0.177
	4	0.635	0.369	0.356	3.251	0.745	0.171
	5	0.639	0.407	0.338	3.078	0.741	0.17
Mah	1	0.612	0.381	0.328	1.68	0.654	0.193
	2	0.6	0.369	0.338	1.595	0.614	0.198
	3	0.589	0.369	0.347	1.858	0.624	0.206
	4	0.61	0.37	0.345	1.614	0.595	0.217
	5	0.638	0.387	0.349	1.616	0.601	0.219
NeGW	1	0.611	0.384	0.33	1.611	0.686	0.19
	2	0.534	0.344	0.38	1.825	0.63	0.202
	3	0.569	0.361	0.369	1.645	0.632	0.206
	4	0.604	0.389	0.337	1.717	0.644	0.2
	5	0.598	0.379	0.354	1.796	0.626	0.207
PsoWE	1	0.381	0.309	0.383	1.101	0.645	0.203
	2	0.404	0.293	0.422	1.351	0.618	0.205
	3	0.481	0.31	0.418	1.244	0.589	0.212
	4	0.504	0.314	0.413	1.202	0.584	0.219
	5	0.475	0.321	0.392	1.249	0.576	0.217

PsoNeGW	1	0.407	0.297	0.412	1.225	0.628	0.199
	2	0.44	0.304	0.42	1.251	0.6	0.228
	3	0.414	0.29	0.44	1.241	0.574	0.219
	4	0.468	0.31	0.417	1.296	0.583	0.221
	5	0.493	0.314	0.419	1.337	0.596	0.221

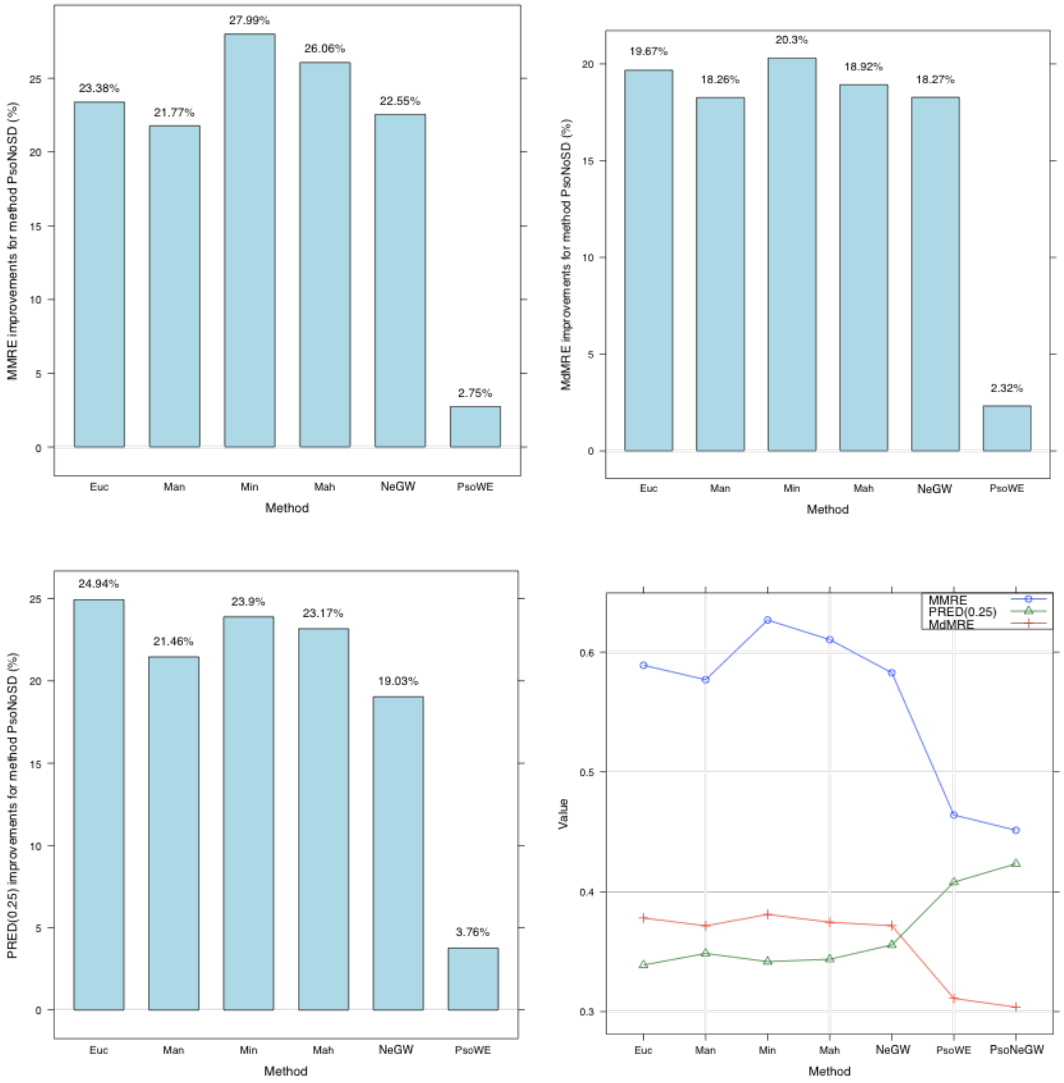


Fig 19. MMRE, MdMRE, PRED(0.25) and Accuracy of PsoNeGW in Desharnais

Figure 18 shows the experimental results on Desharnais dataset. The first three shows the improvement of PsoNeGW method in MMRE, MdMRE and PRED (0.25) and the last line chat shows the difference of MMRE, MdMRE and PRED (0.25) for different methods. As we can see, PsoNeGW method has a significant improvement on the accuracy in all evaluations compare with the other algorithms. Specifically, compared to the Euclidean distance (Euc), Manhattan distance (Man), Minkowski

distance (Min), Mahalanobis distance (Mah) and non-Euclidean geometry weighting (NeGW), PsoNeGW made a very big improvement, on MMRE over 20 % and in PRED (0.25) have a maximum of 24.94% increase. From the results, we can see that PsoNeGW and PsoWE are significantly better than the other methods. Compare with PsoNeGW and PsoWE, PsoNeGW optimized the matrix δ on non-orthogonal space and PsoWE optimized for weight vector, the results show PsoNeGW in MMRE, MdMRE and PRED (0.25) increased 2.75%, 2.32% and 3.76% than PsoWE. This shows that the non-Euclidean geometry weighting method are more accurate in the issue, so by particle swarm optimization algorithm to optimize the non-Euclidean geometry weighting can achieve better estimation accuracy.

In ISBSG R8 dataset, the result of PsoNeGW algorithm is similar to the results of Desharnais dataset (show in Table 11). Compared with other methods, general speaking PsoNeGW made the best estimate results., It is better than the other methods except in MMRE it fell 3.086 percent than PsoWE. Compare with Desharnais dataset, PsoNoSD played a significant role for the improvement in MMRE in ISBSG R8 dataset. Because of the complexity of the dataset, PsoNeGW get smaller effect on the improvement of MdMRE and PRED (0.25). But overall, through the use of non-Euclidean geometry weighting (NeGW), and further use of particle swarm optimization (PsoNeGW), making the software cost estimation accuracy significantly improved.

Table 11. Experiment result of PsoNeGW

Comparable Method	Decrease of $MMRE$	Decrease of $MdMRE$	Increase of $PRED(0.25)$
Euc	38.732%	9.897%	11.585%
Man	38.732%	9.888%	12.239%
Min	58.57%	20.249%	30.294%
Mah	23.777%	3.465%	4.906%
NoSD	25.978%	7.375%	8.423%
PsoWE	−3.086%	0.579%	3.227%

In order to see the effect of different measurement methods and neighbors number K, each method with each K value for MMRE, MdMRE and PRED (0.25) are compared. The compare results are show in Fig 19 and Fig 20.

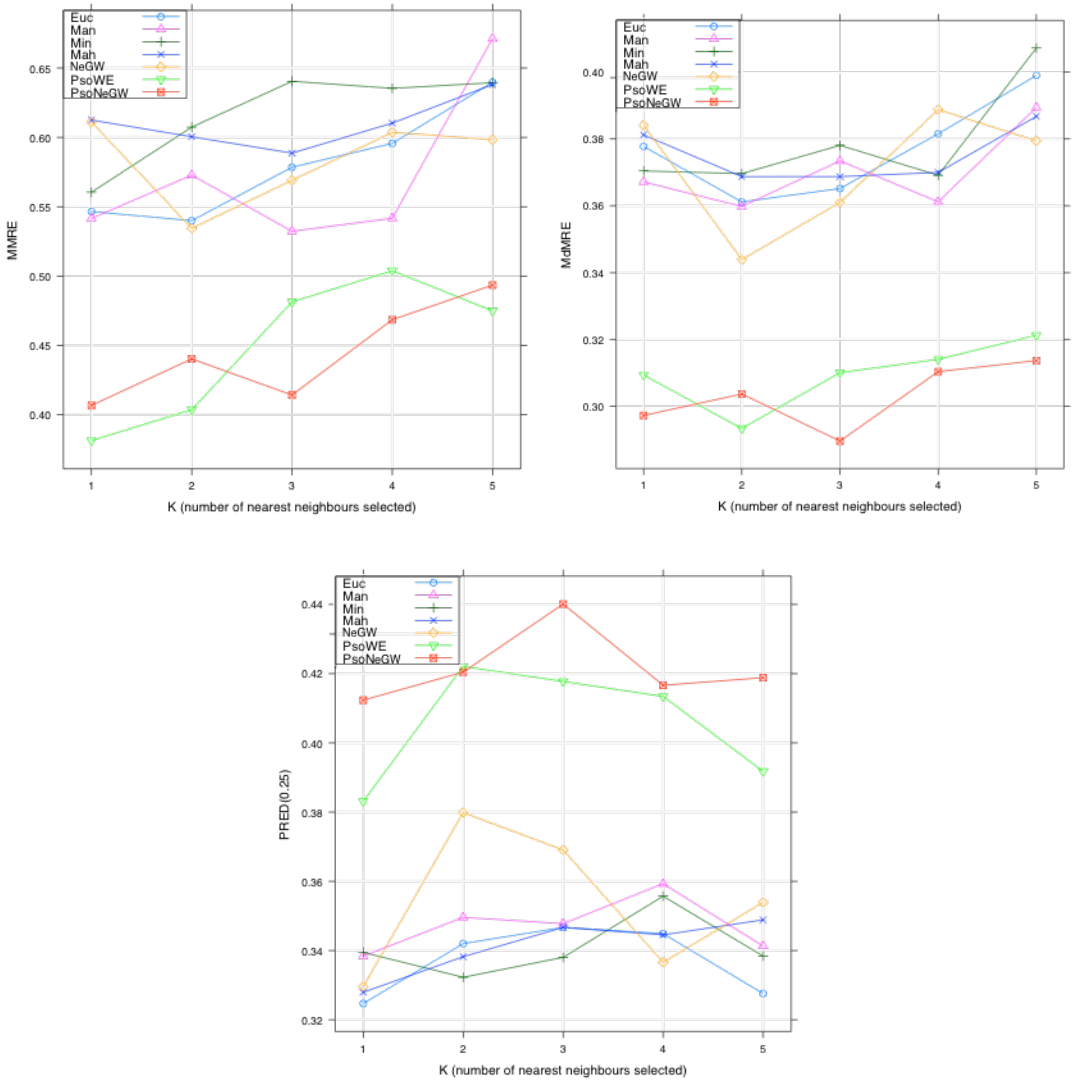


Fig 20. Measurement methods and Neighbors number K comparison in Desharnais

For Desharnais dataset, the prediction accuracy of Euclidean distance (Euc), Manhattan distance (Mah), Minkowski distance (Min) and Mahalanobis distance (Mah) are relatively similar with different neighbors number. Minkowski distance method is a bit less than the other methods. Manhattan distance perform better in MMRE, but similar with the Euclidean distance in PRED (0.25), which is not very stable. When K = 1 or 5, most of the methods appeared to decrease for the accuracy. when K = 2 or 3, the effect of non-Euclidean geometry weighting (NeGW) is excellent, compared to the Euclidean distance, the prediction accuracy is significantly improved. In addition, particle swarm optimization of non-Euclidean geometry weighting (PsoNeGW) method and particle swarm optimization weighted Euclidean distance (PsoWE) had the highest predictive accuracy achieved in all the experimental method. Especially PsoNeGW achieved the best

overall results, compared to the respective other evaluation method has a certain accuracy is improved.

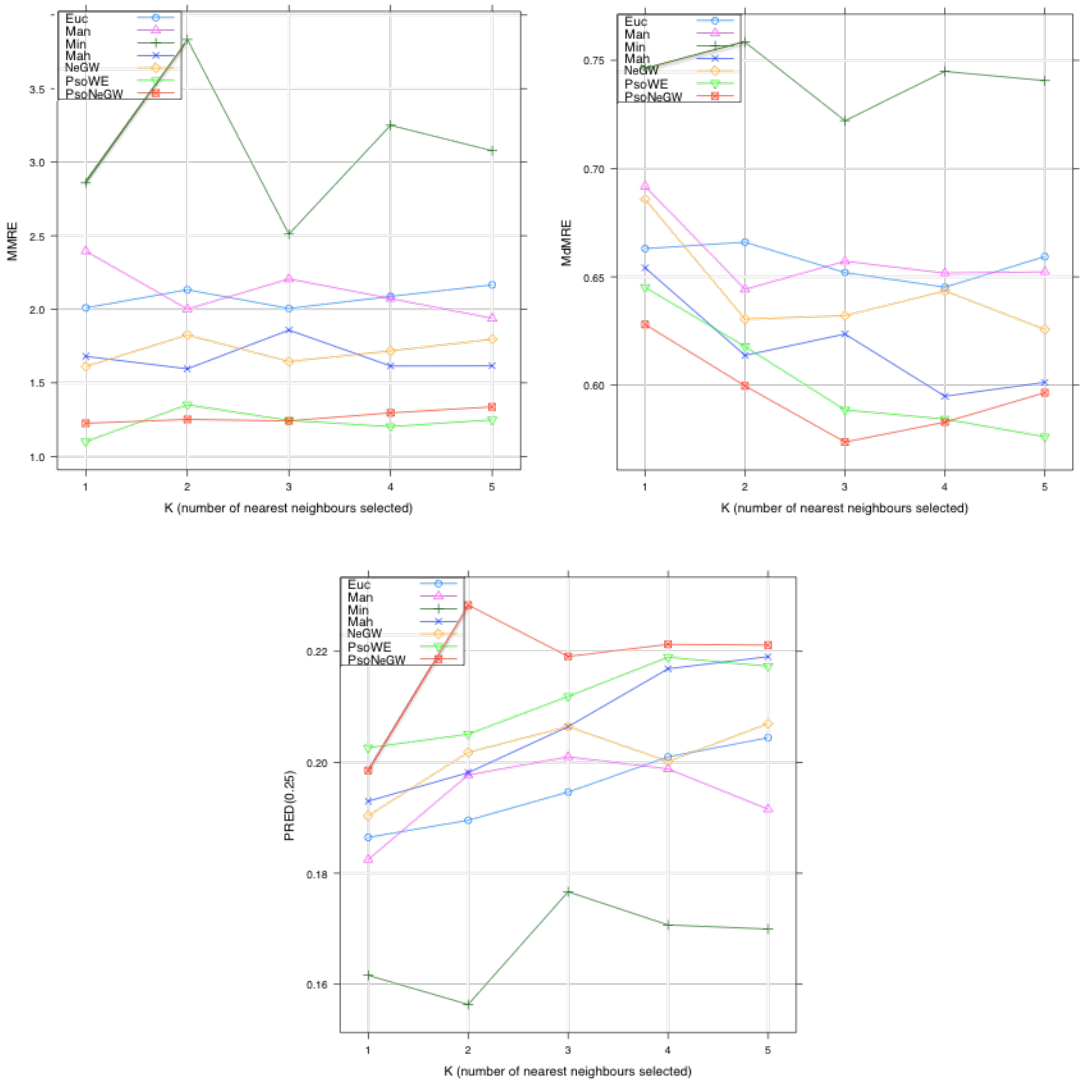


Fig 21. Measurement methods and Neighbors number K comparison in ISBSG R8

The overall results in ISBSG R8 dataset are similar with the result in Desharnais dataset, but there are some differences . First Minkowski distance (Min) in ISBSG R8 dataset is significantly lower than other methods. The reason for it might because ISBSG R8 contains a large number of nominal data, which for distance computing brings many challenges. Also you can see, the Euclidean distance (Euc) and Manhattan distance (Man) is very close, but non-Euclidean geometry weighting (NeGW) method are better than them. Mahalanobis distance (Mah) on ISBSG R8 dataset showed a better prediction accuracy, but with the result in Desharnais dataset, we think NeGW are more stable and perform better than Mahalanobis distance. Same with the result of Desharnais datasets, PsoNeGW

and PsoWE on ISBSG R8 also achieved the best prediction results. PsoNeGW get a bit lower in MMRE than PsoWE, but PRED (0.25) has improved.

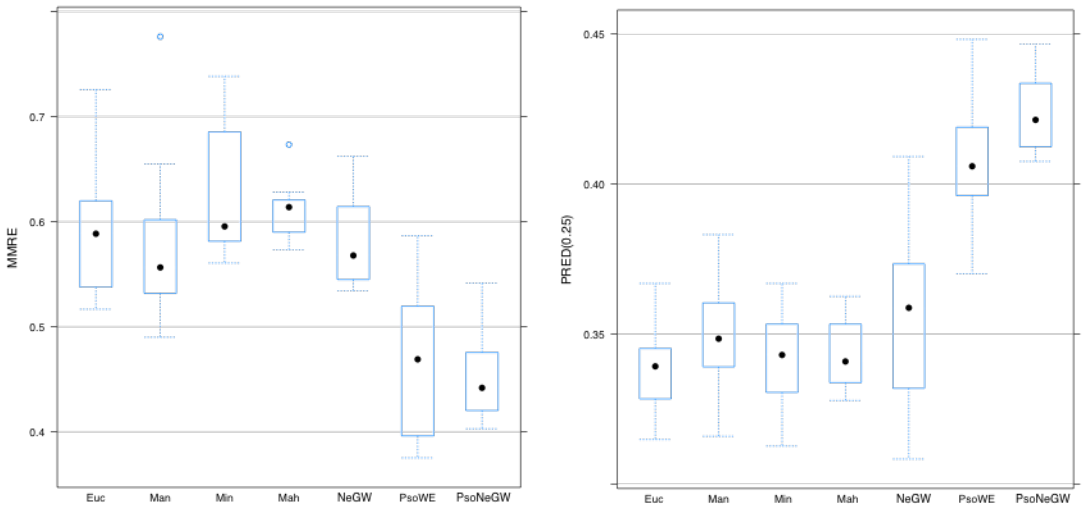


Fig 22. Box plot for MMRE and PRED(0.25) in Desharnais Dataset

Box plot for MMRE and PRED (0.25) results in this two dataset in Fig 21 and Fig 22. From the box plot, how the prediction accuracy of each method is distributed can be roughly found, as well as the stability of each method. In Fig 21, you can see the Euclidean distance (Euc), Manhattan distance (Man), Minkowski distance (Min) are relatively similar in accuracy, Manhattan distance are more stable, and Minkowski distance is poor. The results of Mahalanobis distance (Mah) are relatively similar with Euclidean distance. The non-Euclidean geometry weighting (NeGW) method reaches the best among all the methods. The accuracy of PsoWE and PsoNeGW are significantly higher than other methods, and PsoNeGW are better on the accuracy and stability. The results ISBSG R8 (Fig 22) are similar, but the accuracy of Minkowski distance (Min) is poor. Mahalanobis distance (Mah) and non-Euclidean geometry weighting (NeGW) showed good results. The PsoNoSD and PsoWE prediction accuracy is still the highest and most stable.

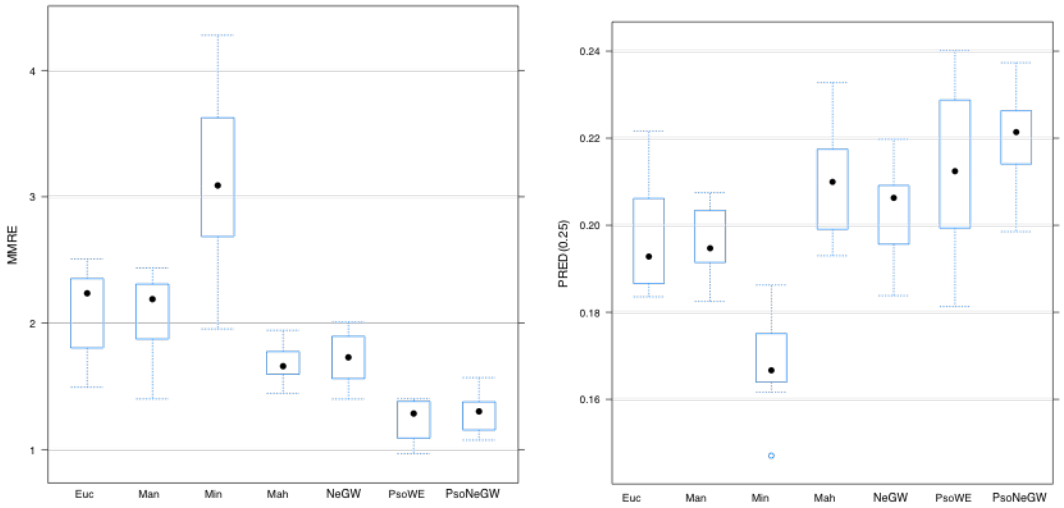


Fig 23. Box plot for MMRE and PRED(0.25) in ISBSG R8 Dataset

Finally, this experiment also compares NeGW-RRD and NeGW-RRQ in non-Euclidean geometry weighting (NeGW) method. The result are shown in Fig 23, generally the results of the two estimates is quite similar, it is difficult to conclude one of the methods will be able to achieve a higher estimation accuracy than the other methods. However, we can see the NeGW-RRD usually more stable than NeGW-RRQ. Although NeGW-RRQ in some cases will achieve better estimation accuracy, but on average, NoSD-RRD are better.

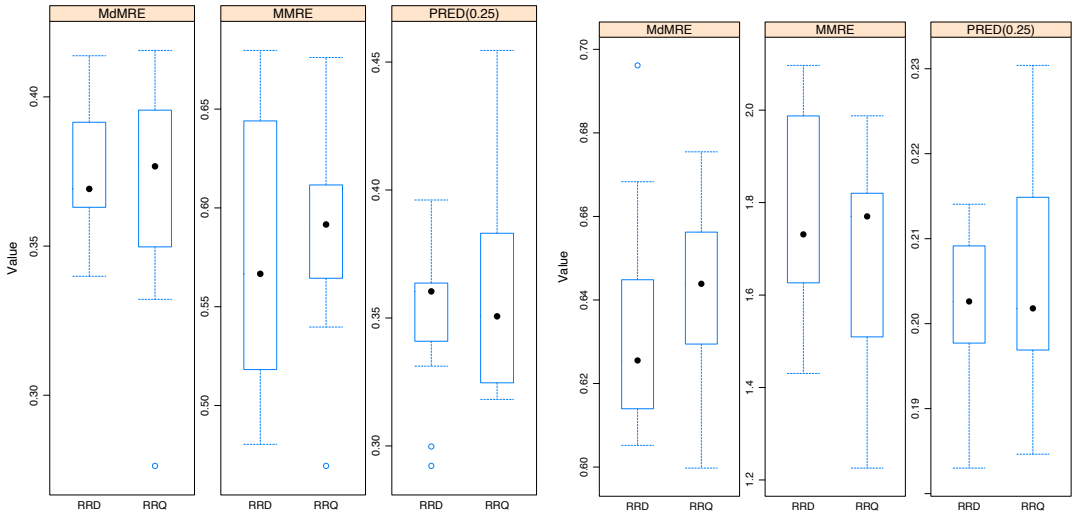


Fig 24. Box plot for MMRE and PRED(0.25) of NeGW-RRD and NeGW-RRQ

6. Verification of the Prototype of the Data Management and Knowledge Clustering System

6.1 Requirement of the pilot application

We are collaborating with an international online payment company. In this company, they suffer a lot from fraudulent transactions, which result in the loss of customers and service providers.

In common scenarios, they found those transactions occur with patterns:

- One account is hacked into by criminals, then multiple fraudulent transactions are submitted to transfer money from that account to somewhere else.
- Illegal accounts are created and linked to a stolen (or fake) credit card, and then fraudulent transactions are submitted to withdraw money from those accounts.
- Customers' computers are affected by Special designed virus, and then multiple fraudulent transactions are submitted to transfer their money into one account.

Every pattern can be seen as a set of accounts and transactions, where those transactions or accounts have strong relations with each other. (e.g. Similar transaction time, Same IP Address, and so forth).

In order to make better prediction of the fraud transaction, if a transaction graph (or network) based on the relationships of accounts and transactions can be built. So all the historical data can be linked to generate the universal transaction graph. By importing the social network dataset, a better understanding about the users on that account can be built. By importing the anti-various company's dataset, a better understand about the stories can be found

behind the IP address. By adapting graph analysis techniques, fraudulent networks can be identified, which exist as sub-graphs of the universal transaction graph and cluster those fraudulent networks and generate fraudulent transaction patterns. Finally, with all the models and information above, an unknown transaction can be recognized whether is fraudulent or not.

With the support of the several companies, a pilot project of data management and knowledge sharing system is built in fraud detection. The idea behind the pilot project is that if more information or more data from different organizations and individuals about the user and transaction environment (IP etc.) can be merged, the better decision can be made to predict whether it is a fraud transaction or not. We involve an online payment company A, an e-business company B, an Anti-virus company C and a social network company D (for confidential reason, use A, B, C and D to represent real name). Company A shares several online payment datasets, which includes account information and transaction information. Company B shares several business transaction datasets, which include the payment account and user information, product information. Company C shares a dataset of risky IP address; Company D shares a dataset of the linkage of related account in social network.

Because the dataset they offered are all in large volume (Total volume over 100 Terabytes), and it is impossible to deal with them in a single computer. They datasets contain different information about the user, transaction, environment and items, all these are related with each other; they will naturally form a heterogeneous graph, we can analysis the risk transactions based on the heterogeneous user-transaction-environment graph.

6.2 Application design for the Pilot system

The physical architecture of the whole system will like figure 24. There is one master server node in the system, which will in charge of data management and server management; several slave servers which will be used for storage the dataset, update, delete and search services. All the slave servers can be in the same subnet, which is connected by routers, exchangers.

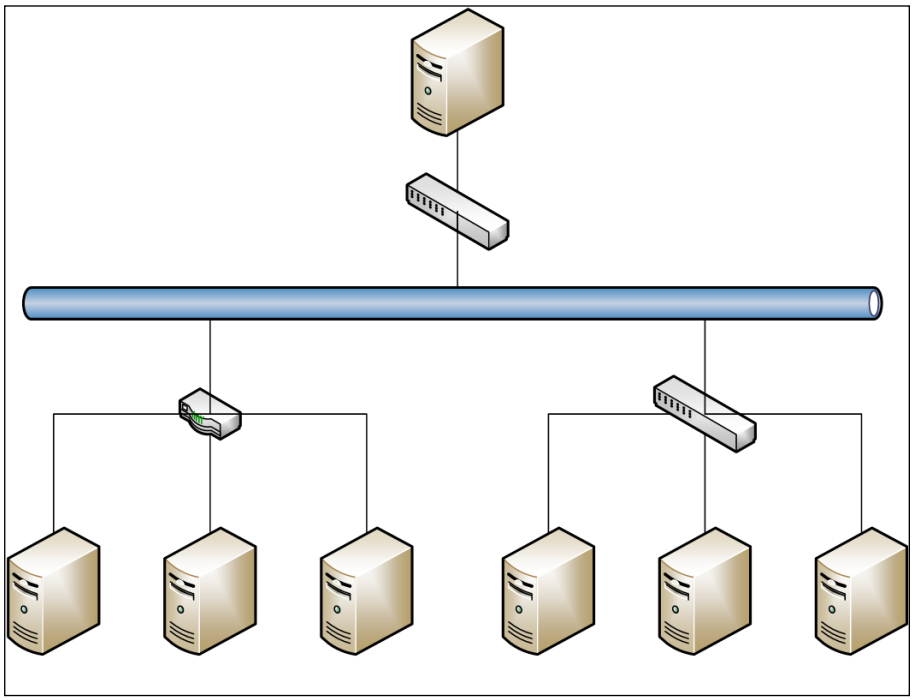


Fig 25. Physical deployment of the system

With different facilities, master server and slave server can in the same subnet like finger 25.

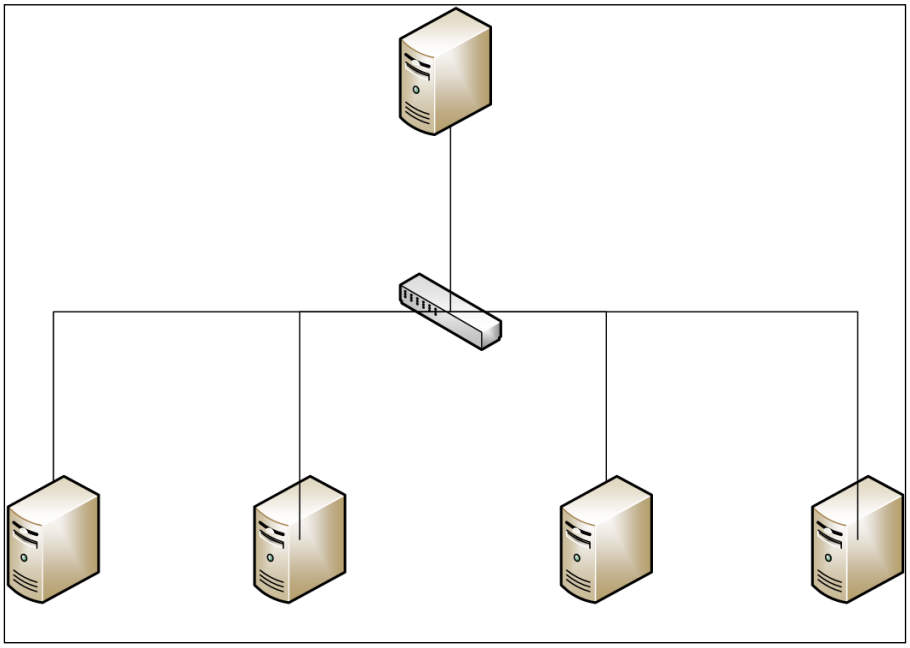


Fig 26. Slaver Server in the same network

The pilot project cluster consists of one master node and multiple slaver service nodes. The pilot project will be run as a user-level program on Linux. In this pilot project, different datasets will be merged and build heterogeneous graph (which contain data from multiple vertex and edge). All the dataset and

heterogeneous graph will be stored in the indexed HDFS system as proposed in Chapter 4, and will be redundant coped according to the policy of HDFS blocks to ensure that data is not lost with the failure of a single node. We will use the algorithms as proposed in Chapter 5 to predict the fraud transaction. Since this is an internal project between these four companies, the IP protection and serious game idea as proposed in Chapter 6 will not be involved. The whole system architecture is show in Figure 26.

The main functions of the master server are: management and maintenance of the slaver servers, including adding new and deleting servers; monitoring the server statements; management and maintenance graph storage, including graph data assignment, migration, etc.; manage and maintain the index server and the index server assignment, transfer, etc.

The main functions of the slaver server are: responsible for storing the dataset and response for data read and write; running the analysis algorithm for fraud detection

The main functions of the index server are: the index server is elected from the slave servers in the same network, response for check their IP of slave node from the graph vertex ID.

In order to effectively control the servers, zookeeper module is used. This module can make good monitoring of adding and deleting of servers and will feedback to the master node to dynamically maintain the whole system.

The system uses the RPC protocol to establish communication with the master server from the slaver servers. When the system is ready for operation, the master node will automatically find the minimum load balanced slaver server to response fro the operation, which will improve the efficiency of the resource usage. Specifically, when you insert a vertex or edge, the master node will assess a requesting of counting evaluation score to each slaver server. The highest score one will be selected to store. The evaluation is:

$$Mark = topoMark - vCount * 0.01 - eCount * 0.01 \quad \text{Eq. 20}$$

Where:

$Mark$: evaluation scores, the higher the more suitable to store

$topoMark$: topological structure evaluation score (the number of connections to the node stored in the server), the higher the more representative with this

server

vCount : total number of vertices in the server, represent the load balance of the server

eCount : total number of edges in the server, represent the load balance of the server

In order to reduce the total running jobs and accelerate the query performance, we will pickup one slaver server to be index server. To avoid exceeding the process capacity dynamically, the master server will monitor the load balance of the index server, if the load balance exceeds a certain range, the master server will switch index into another server. The server will count the index server's load balance as following:

$$usageMark = W_c \times \text{Usage}_{CPU} + W_m \times \text{Usage}_{MEM} \quad \text{Eq. 21}$$

Where:

usageMark : evaluation scores , the higher the higher the load balance on this server

W_c : Weight of CPU occupation

W_m : Weight of Memory occupation

Usage_{CPU} : CPU usage ratio

Usage_{MEM} : Memory usage ratio

When storing dataset into the system and building the heterogeneous graph, a priority number is given to each attribute, which can be indexed. When the system runs, the priority also counted based the statistic of queries (frequently queried data will be given a higher priority), and then collect the statistics of cache hit rate as well. After Combine these three indicators, the system was pick up the most proper attributes and put the index into memory, the priority will be calculated like this:

$$Cache\_priority = A_1X_1 + A_2X_2 + A_3X_3 \quad \text{Eq. 22}$$

Where :

$Cache\_priority$: Cache priority ;

X_1 : user-specified priority ;

X_2 : The number of recent searches ;

X_3 : The cache miss ratio ;

A_1, A_2, A_3 : weights of the indicators, $A_1 + A_2 + A_3 = 1$

After calculating the delay priority, the buffer size of the data is:

$$Cache\_size = \frac{Cache\_priority}{SUM(Cache\_priority)} \times MaxCacheSize \qquad \text{Eq. 23}$$

Where :

$Cache\_size$: Buffer size for the cache;

$MaxCacheSize$: A total size in the server for cache;

6.3 Data Model for the Pilot system

The pilot system will build heterogeneous graph based on the dataset. In heterogeneous, there are two part to discuss, first is topology data structure for graph, which include node, edge, attributes and linkage; second is the datasets from different source. The pilot system is divided the data model into two parts: graph layer and the dataset layer, which will store the preview two part. The layered data mode is shown below:

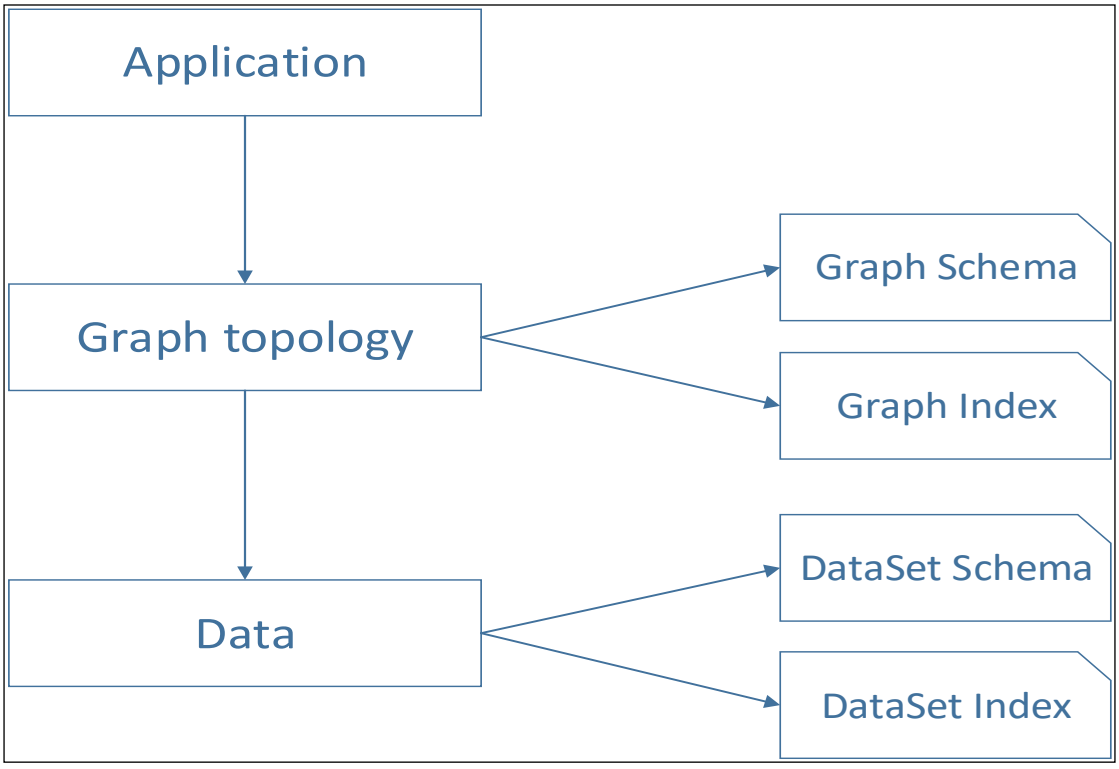


Fig 27. Layered data model in the pilot system

As shown in Figure 27, the pilot system needs to store and processed heterogeneous data. In the layered data model: 1) different kinds of vertex and edge in the heterogeneous graph storage system, the attribute of different kinds of

vertex and edge are different; 2) dataset can be stored with different format. In the layered data models in the pilot system, on graph topology and data layer, all have schema and index, which describe data structure in each layer and the state of index. We will introduce each layer in the data model:

(1) Graph topology layer

In graph topology layer, there are three different objects: Vertex, Edge and Schema. The properties of these three objects are as follows:

Table 12 Property of Vertex object

Property	Data type	Comment
ID	java.lang.String	ID of the vertex, globally unique, will be store: V_XXXX (V represent Vertex)
Edge_List	java.util.LinkedList	List of all the edges started with a vertex, each item in the list is: <Edge_id, Target_Vertex_id>, which represent edge ID and target vertex ID
Schema_id	java.lang.String	Point to the ID of Schema Table, which represent the type of the vertex
Graph_id	java.lang.String	Point to the ID of Graph
Pointer_List	java.util.HashMap	Store the relation of the attribute of the vertex and the dataset. The relation wil be stored as a list: <attri_name,source_pointer_list>, atrti_name is the name of the attribute and source_pointer_list point to a certain line of dataset schema in data layer.

Table 13. Property of Edge object

Property	Data Type	Comment
ID	java.lang.String	ID of the edge, globally unique, will be store: E_XXXX (E represent Edge)
Source_Vertex_id	java.lang.String	ID of source vertex

Target_Vertex_id	java.lang.String	ID of target vertex
Schema_id	java.lang.String	Point to the ID of Schema Table, which represent the type of the edge
Pointer_List	java.util.HashMap	Store the relation of the attribute of the edge and the dataset. The relation wil be stored as a list: <attri_name,source_pointer_list>, atrti_name is the name of the attribute and source_pointer_list point to a certain line of dataset schema in data layer.

Table 14. Property of Schema object

Property	Data Type	Comment
sId	java.lang.String	ID of the graph schema, globally unique, will be store: GS_XXXX (GS represent Graph Schema)
Attributes	java.util.ArrayList	The desctiption of graph attribute, each item will be stored: <Name,DataType>, which Name represent the name of the attribute, Data Type represent type, such as Int, Long, char(20), etc.

(2) Data layer

a) Architecture

The index mechanisms are an optimized implementation of traditional MapReduce query and process. Because MapReduce does not support for index, it will scans all the blocks of the dataset parallel. The pilot system will combine index and Schema mechanisms to reduce bandwidth consumption, reduce the total jobs running in the system and optimize the query to find efficiencies in most cases.

The data layer consists of a Master server and N slaver servers. All the Schema

files are stored and managed by the Zookeeper. Each rack contain a cache for index, frequency used index will be cached in memory of a selected server in the rack. The selected server will be elected by the electoral mechanism. The other index file will be store in the HDFS system.

b) Structure of Schema and the storage of Schema

- Structure of Schema

Each schema will have a unique SchemaID, which is global unique ID. Since multi schema can be given for an dataset, the end user has the right to choose which schema they want use to describe the dataset.

The logical structure of Schema will listed as below:

SchemaID: Unique ID of Schema.

Separator: the separator in the row of dataset, could be a comma, and could be no separators.

Column Description: the description of the attribute in the dataset, which includes:

Name: Name of the attribute

Range: Related with Separator, if there is a separator, range will mark which column separated by separator belongs to the attribute in the dataset; if there is no separator, range will be from X bit to Y bit in the dataset.

Flag: This marks whether this column is needed to be indexed in the system. Not all the attributes are suitable for creating index, only those frequent queried attribute will do.

- Storage of Schema

All the schema files will be stored in the Zookeeper server.

c) Structure of Index and the storage of Index

- Structure of Index

Index will be stored in the binary form: <Key, Value>, key is the id of index, which is global unique and identifies the index. IndexId will be stored in the form: "DatasetId + SchemaId + ColumnName". From the definition, each column in a dataset under a Schema may have a unique index. The system will build the B+ tree for this column data when it starts building the index. In the B+ tree node, BlockID and LineNumber will be stored in the node, after serialization, system will store the B+ tree into the value part.

The initial state of the system is no index, all the indexes will be established when it is queried.

- Storage of Index

After serialization, index will be stored as a file of the binary form. There is a cache buffer in each rack; the most frequently used index will be saved in the memory of a certain server. This server is selected by the electoral mechanism. The others index will be saved as files in HDFS.

c) Data Flow

- Schema Maintenance

Schema will be manually added and updated.

- Index Maintenance

There will be no index file at the initial state of the system, the index will be only created when the column is queried by the system (except Schema Flag marked as no index). The index will be stored in cache to gain better performance. When the cache is full, system will maintain and update the cache.

The whole system will set the number of query times M, when the cache is full, it will automatically check for nearest M times query, the index of N high-frequency queried column will stay remain in the cache, the rest of index will be moved to the HDFS file system.

- Single target query process

For single target query, the index system will not be used, there is no difference between original Hadoop system and our pilot system. System will receive a query with these parameters: DatasetID, BlockID, LineNumber and SchemaID. The execution server will find the block with the BlockID, then use the LineNumber to read particular LineNumber row, with the SchemaID, it can understand the dataset and return the result.

- Multiple targets query process

There are two typical queries in the system:

1. DatasetID, SchemaID, a certain attribute = value

2. DatasetId, SchemaID, a certain attribute value in a range

According to DatasetID, SchemaID and attribute information, we can check out whether the attribute is indexed, if it is indexed, the indexed query process will be followed and otherwise, the un-indexed query process will be followed.

- Indexed query process

Step 1. Find the index in Cache, if founded, goto Step 2, otherwise continue to search in the HDFS system for the index. If it is available in HDFS, goto Step 2. If still no result, build the index for this column, serialize it and stored in the cache.

Step 2. Scan the index; find out the LineNumber and BlockID for the result.

Step 3. Use the improved MapReduce to obtain query results, before Map job start, the slicing is filtered to only keep the block in Step 2.

Step 4. If it is needed to return the value of an attribute, the MapReduce job will filter according to Schema. At the end of the MapReduce job, the query results will be available.

- Un-indexed query process

The query process will follow the standard MapReduce process. If it is needed to return the value of an attribute, the MapReduce job will filter according to Schema. At the end of the MapReduce job, the query results will be available.

6.4 Data Access Interface for the Pilot system

The data access interface of master server is listed in Figure and Table

```
public interface GMasterProtocol extends VersionedProtocol {

    //For Graph

        //Graph Data

            //insert
            public String findTargetGServer_Store(VertexInfo infomation);
            public String findTargetGServer_StoreEdge(EdgeInfo information);
            //update
            //remove
            //query

        //Graph Index

            //insert
            public void insertVertexInfoToIndex(String vid, String ip);
            public void insertEdgeInfoToIndex(String eid, String ip);
            //update
            //remove
            public void removeVertexFromIndex(String vid);
            public void removeEdgeFromIndex(String eid);
            //query

        //Graph Schema

    //End of Graph
```

```
//For DataSet

    //DataSet PathIndex

        //insertOrUpdate
        public void notifyDataSet_Insert(String source, String dsID, String
        //remove
        public void notifyDataSet_Remove(String source, String dsID);

    //DataSet Index

        //remove
        public void notifyDataSet_Index_Remove(String source, String dsID,

//End of DataSet


//For Server Management

    //For Index Server

        //insert
        //update
        public void requestToChangeIndexServer(String source_ip);
        //remove
        //query

//End of Server Management

public void stopService();
```

Table 15. Data access interface of master server

Interface name and Function Description	Parameter	Return Value
findTargetGServer_Store (Find out the most suitable slaver server to store the vertex based on the overall load balance and topologic structure)	VertexInfo Object, represent for the vertex to be inserted	String, the IP of the slaver server
findTargetGServer_StoreEdge (Find out the most suitable slaver server to store the edge based on the overall load balance and topologic structure)	EdgeInfo Object, represent for the edge to be inserted	String, the IP of the slaver server
insertVertexInfoToIndex (Insert the vertex information into	vid: New Vertex ID ip: IP of slaver server	None

index)		
removeVertexFromIndex (Delete the vertex information from index)	vid：the Vertex ID needed to be delete	None
insertEdgeInfoToIndex (Insert the edge information into index)	eid：New Edge ID ip: IP of slaver server	None
removeEdgeFromIndex (Delete the edge information from index)	eid：the Edge ID needed to be delete	None
notifyDataSet_Insert (Broadcast the insert of a dataset to all the index server, maintain the consistency of the index metadata)	source: slave server IP to deal with insert. dsID: Dataset ID that need to be insert hdfsPath: HDFS path for the dataset	None
notifyDataSet_Remove (Broadcast the delete of a dataset to all the index server, maintain the consistency of the index metadata)	source: slave server IP to deal with delete. dsID: Dataset ID that need to be delete	None
notifyDataSet_Index_Remove (Broadcast the delete of an index to all the index server, maintain the consistency of the index metadata)	source: slave server IP to deal with delete. dsID: Dataset ID that need to be delete dschemaID: Data Schema ID that need to be delete attriName: Attribute Name that need to be delete	None
requestToChangeIndexServer (Change the index server)	source_ip：slave server IP to deal with request.	None
stopService	None	None

(Stop Service)

The data access interface of slave server is listed in Figure and Table

```
public interface GServerProtocol extends VersionedProtocol {

//For Graph

    //Graph Data

        //insert
        public String storeVertex(VertexInfo vdata, EdgeCollectionWritable edat
        public String storeEdge(EdgeInfo edata);
        public String storeVertexList(VertexCollectionWritable vdata,
                EdgeCollectionWritable edata);
        public float getMarkForTargetVertex(VertexInfo info);
        //update
        public boolean updateVertexInfo(VertexInfo info);
        public boolean updateVertexInfo_Remote(VertexInfo info);
        //remove
        public boolean removeVertex(String id);
        public boolean removeVertex_Remote(String id);
        public boolean removeEdge(String id, String source_vertex_id);
        public boolean removeEdge_Remote(String id, String source_vertex_id);
        //query
        public VertexInfo getVertexInfo(String id);
        public VertexInfo getVertexInfo_Remote(String id);
        public VertexData getVertexData(String id);
        public EdgeInfo getEdgeInfo(String id);
        public EdgeData getEdgeData(String id);
```

```
    //Graph Index

        //insert | update
        public void putVertexInfoToIndex(String vid, String targetIP);
        public void putEdgeInfoToIndex(String eid, String targetIP);
        public void putVListToIndex(StringMapWritable map);
        public void putEListToIndex(StringMapWritable map);
        //remove
        public void deleteVertexFromIndex(String vid);
        public void deleteEdgeFromIndex(String eid);
        //query
        public String queryVertexToServer(String vid);
        public String queryEdgeToServer(String eid);
        //manage
        public double reportUsageMark();
        public void assignIndexServer(BPlusTreeStrStrWritable vertexIndex, BPlu
        public void announceIndexServer(String ip);

    //Graph Schema

        //insert | update
        public boolean insertOrUpdateSchema(String graphid, Graph_Schema gs);
        //remove
        public boolean removeSchema(String graph_id, String schema_id);
        //query

//End of Graph
```

```
//For DataSet Management

    // DataSet (Management by Zookeeper)

        //insert | NOUpdate
        public boolean insertDataSet(String dsID, String hdfsPath);
        public boolean insertDataSet_Sync(String dsID, String hdfsPath);
        //remove
        public boolean removeDataSet(String dsID);
        public boolean removeDataSet_Sync(String dsID);
        //query
        public String getDataSetPath(String dsID);
        public String getDataSetPath_Remote(String dsID);

    // DataSet Index

        //create (Commit a MapReduce Job to analysis the dataset and create a b
        public String createDSIndex(String dsID, String dschemaID, String attri
        //update (remove and re-create)
        public String updateDSIndex(String dsID, String dschemaID, String attri
        //remove (remove HDFS file, update the cache)
        public String removeDSIndex(String dsID, String dschemaID, String attri
        public void removeDSIndex_Sync(String dsID, String dschemaID, String at
        //manage
        public BPlusTreeStrLongWritable getDSIndex(String dsID, String dschemaI
```

```
    // DataSet Schema

        //insert | update
        public boolean insertOrUpdateDataSchema(String dschemaID, Data_Schema d
        //remove
        public boolean removeDataSchema(String dschemaID);
        //query
        public Data_Schema getDataSchema(String dschemaID);

//End of DataSet


//For Server Management
public void stopService();
//End of Server Management
```

Table 16. Data access interface of slave server

Interface name and Function Description	Parameter	Return Value
storeVertex (Store Vertex in the system)	VertexInfo: vertex information EdgeCollection: edges connected with the vertex	String for error information, NULL for success
storeEdge	EdgeInfo: edge	String for error

(Store Edge in the system)	information	information, NULL for success
storeVertexList **(Batch store vertex in the system)**	VertexCollection: a set of vertex EdgeCollection：edges connected with the set of vertex	String for error information, NULL for success
getMarkForTargetVertex **(Evalute whether it is suitable to store the vertex locally)**	VertexInfo：vertex information for evaluation	float：evaluation score, the higher the suitable to store locally
updateVertexInfo **(Update the local vertex information)**	VertexInfo: vertex information for updating	Boolean：whether the update successes
updateVertexInfo_Remote **(Update the vertex information in the system)**	VertexInfo: vertex information for updating	Boolean：whether the update successes
removeVertex **(Delete local vertex)**	id: vertex id for the remove	Boolean：whether the remove successes
removeVertex_Remote **(Delete vertex in the system)**	id: vertex id for the remove	Boolean：whether the remove successes
removeEdge **(Delete local edge)**	id: edge id for the remove source_vertex_id: vertex id of the edge	Boolean：whether the remove successes
removeEdge_Remote **(Delete edge in the system)**	id: edge id for the remove source_vertex_id: vertex id of the edge	Boolean：whether the remove successes

getVertexInfo **(Read the local Vertex information)**	id： vertex id for reading	VertexInfo
getVertexInfo_Remote **(read the vertex information in the system)**	id： vertex id for reading	VertexInfo
getVertexData **(Read the vertex Data in the dataset)**	id： vertex id for reading	VertexData
getEdgeInfo **(Read the local Edge information)**	id： edge id for reading	EdgeInfo
getEdgeData **(Read the edge Data in the dataset)**	id： edge id for reading	EdgeData
reportUsageMark **(Evaluate the load balance in the system)**	none	double：score of the evaluation
assignIndexServer **(Set current server to be the index server)**	none	none
announceIndexServer **(Broadcast the index Server)**	ip：the IP of index server	none
putVertexInfoToIndex **(Insert vertex information into the index system)**	vid： vertex id for insert targetip：the server IP that store the vertex	none
putEdgeInfoToIndex **(Insert edge information into the index system)**	eid： edge id for insert targetip：the server IP that store the	none

	edge	
putVListToIndex **(Insert a set of vertexes into the index system)**	StringMapWritable：a set of vertex	none
putEListToIndex **(Insert a set of edges into the index system)**	StringMapWritable：a set of Edge	none
deleteVertexFromIndex **(Delete vertex from index system)**	vid：vertex id that will be removed from index system	none
deleteEdgeFromIndex **(Delete edge from index system)**	eid：edge id that will be removed from index system	none
queryVertexToServer **(Obtain the server IP of a vertex)**	vid：vertex id that will be queried from index system	The ip of the server
queryEdgeToServer **(Obtain the server IP of a edge)**	eid：edge id that will be queried from index system	The ip of the server
insertOrUpdateSchema **(Insert or Update of the schema)**	graphid: graph id for insert gs: Graph_Schema fro insert	Boolean：whether the operation successes
removeSchema **(Remove the schema)**	graphid: graph ID schema_id: Graph_Schema ID	Boolean：whether the operation successes
insertDataSet **(Insert dataset and notify master server)**	dsID: dataset ID hdfsPath：path in HDFS	Boolean：whether the operation successes
insertDataSet_Sync **(Insert dataset and update)**	dsID: dataset ID hdfsPath：path in HDFS	Boolean：whether the operation successes

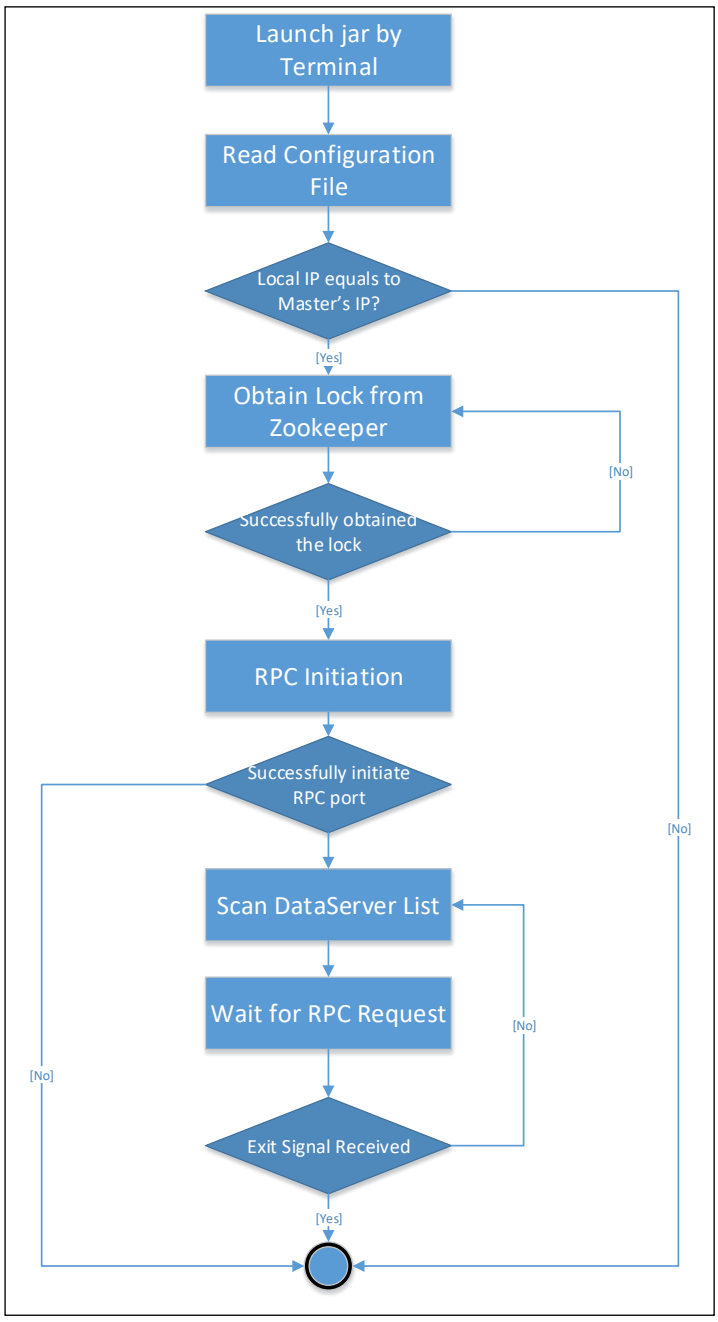
removeDataSet **(Delete dataset and notify master server)**	dsID: dataset ID	Boolean：whether the operation successes
removeDataSet_Sync **(Delete dataset and update)**	dsID: dataset ID	Boolean：whether the operation successes
getDataSetPath **(Get the local dataset path)**	dsID: dataset ID	String: the path of the dataset
getDataSetPath_Remote **(Get the dataset path in the system)**	dsID: dataset ID	String: the path of the dataset
createDSIndex **(Create dataset index)**	dsID: dataset id for create dschemaID: related SchemaID for index attriName: column name of attribute	Boolean：whether the operation successes
updateDSIndex **(Update dataset index)**	dsID: dataset id for update dschemaID: related SchemaID for index attriName: column name of attribute	Boolean：whether the operation successes
removeDSIndex **(Remove dataset index and notify the master server)**	dsID: dataset id for delete dschemaID: related SchemaID for index attriName: column name of attribute	Boolean：whether the operation successes
removeDSIndex_Sync **(Remove dataset index and update the cache)**	dsID: dataset id for delete dschemaID: related	Boolean：whether the operation successes

	SchemaID for index attriName: column name of attribute	
getDSIndex **(Obtain dataset index)**	dsID: dataset id for obtain dschemaID: related SchemaID for index attriName: column name of attribute	BPlusTreeStrLong Writable: the index in B+ tree format
insertOrUpdateDataSchema **(Insert or update the dataset schema)**	graphid: id of graph for insert ds: Data_Schema object for insert	Boolean：whether the operation successes
removeDataSchema **(Delete dataset schema)**	dschemaID: id of schema for delete	Boolean：whether the operation successes
getDataSchema **(Obtain dataset schma)**	dschemaID: id of schema for obtain	Data_Schema
stopService **(Stop the whole system)**	none	none

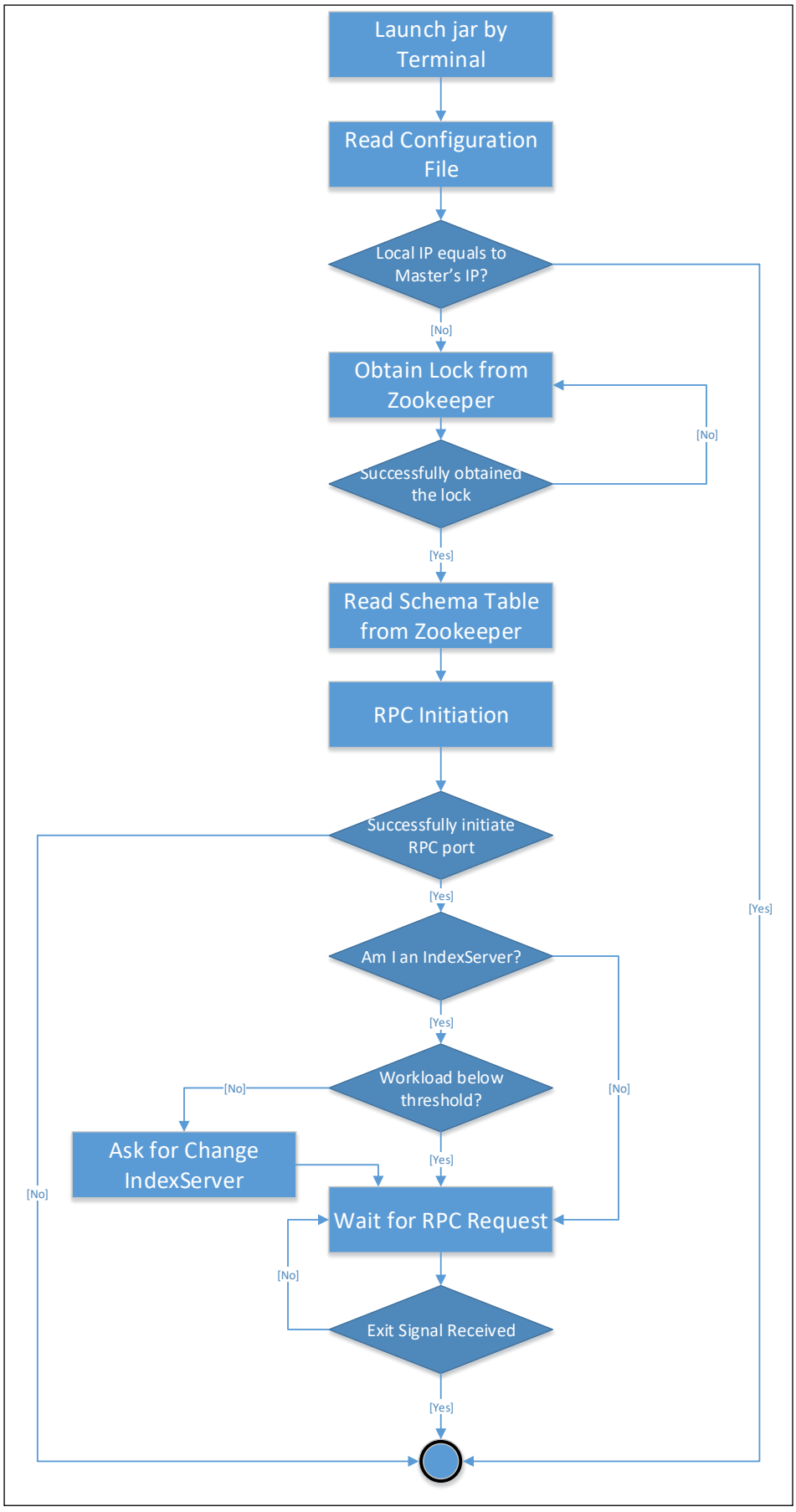
6.5 The implementation for the Pilot system

We will show some figure about the importance workflows in the pilot system:

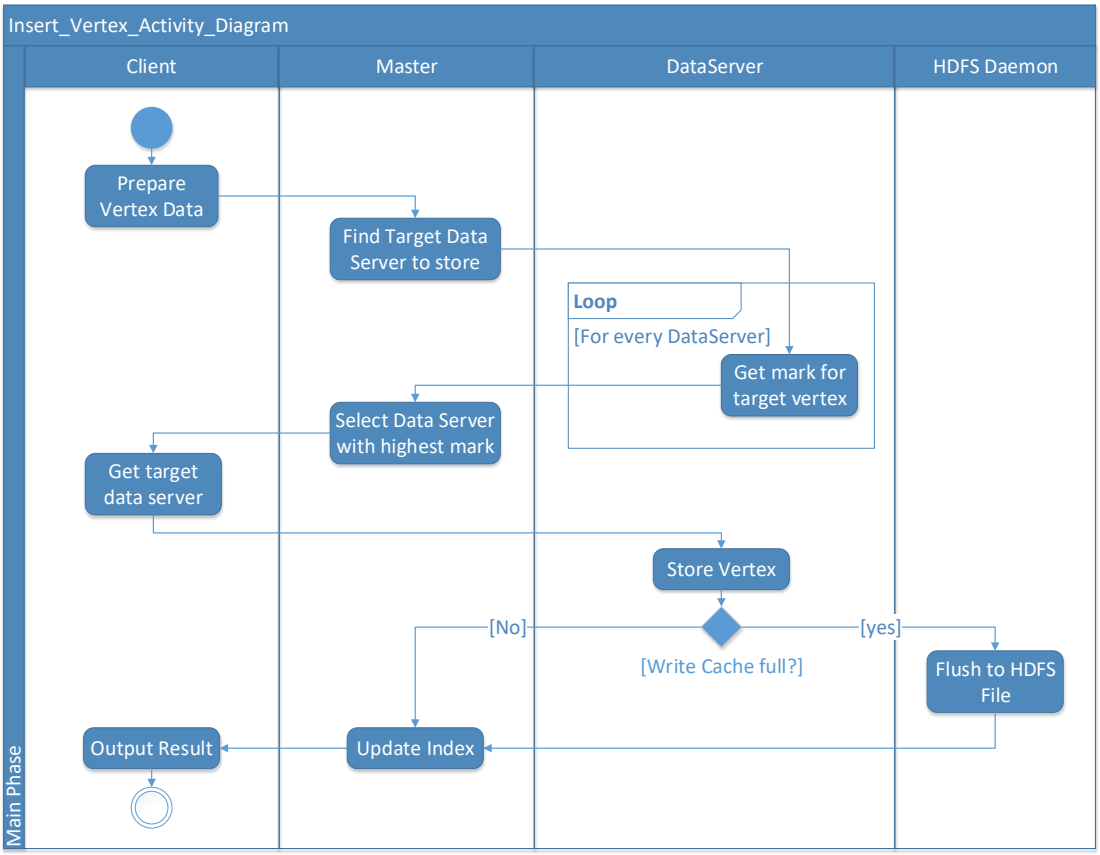
- The start workflow for master server:



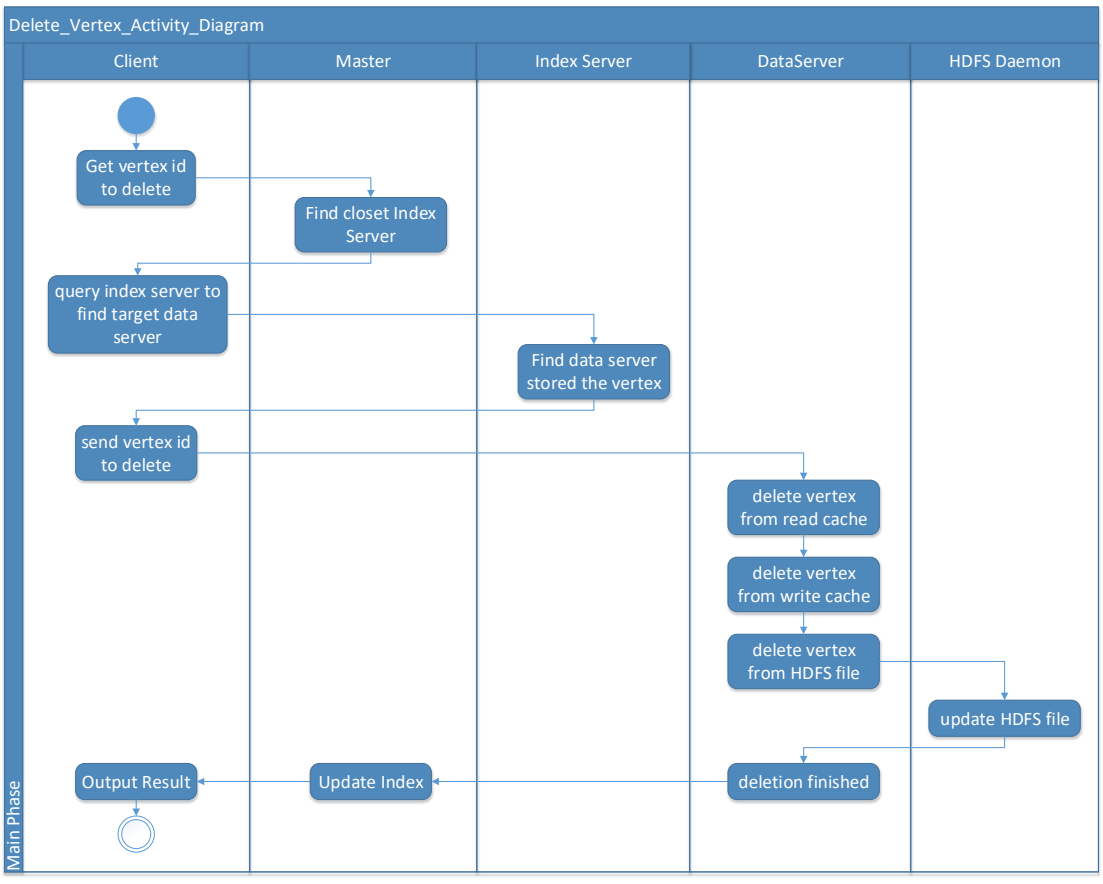
- The start workflow for slave server:



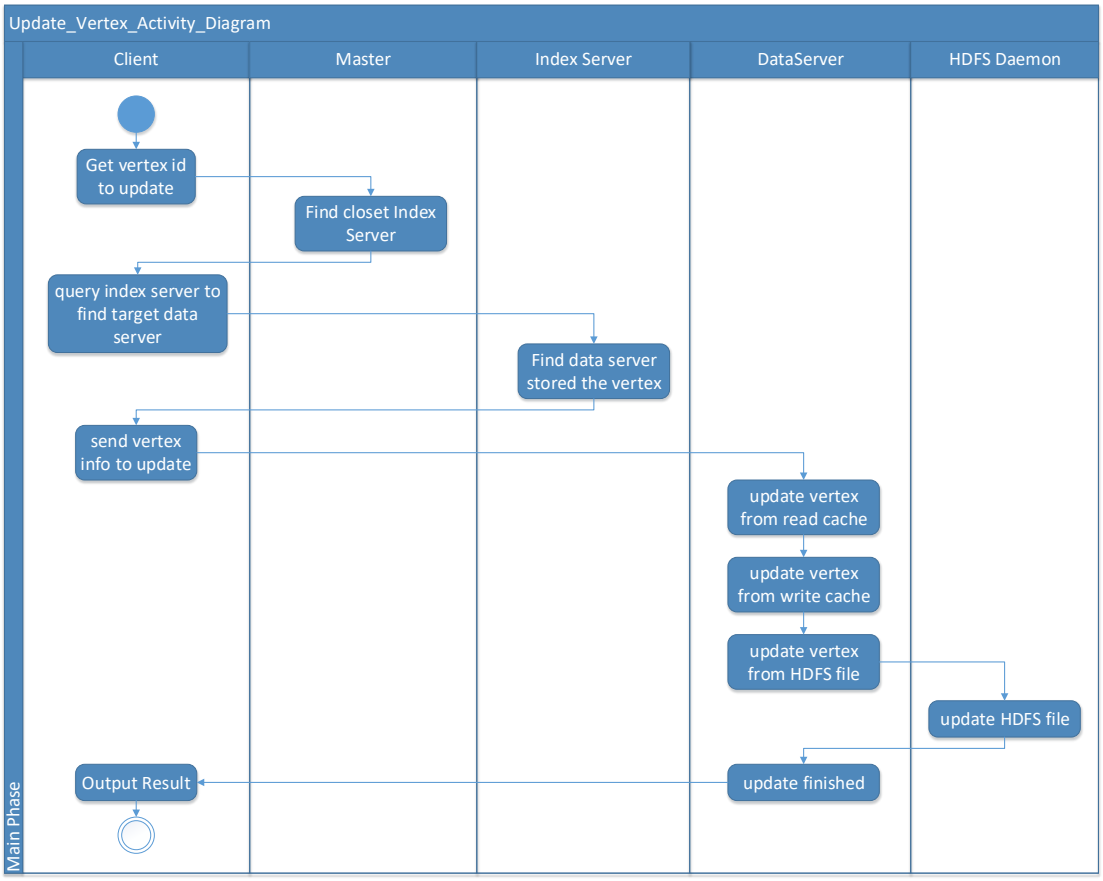
- The workflow of inserting vertex:



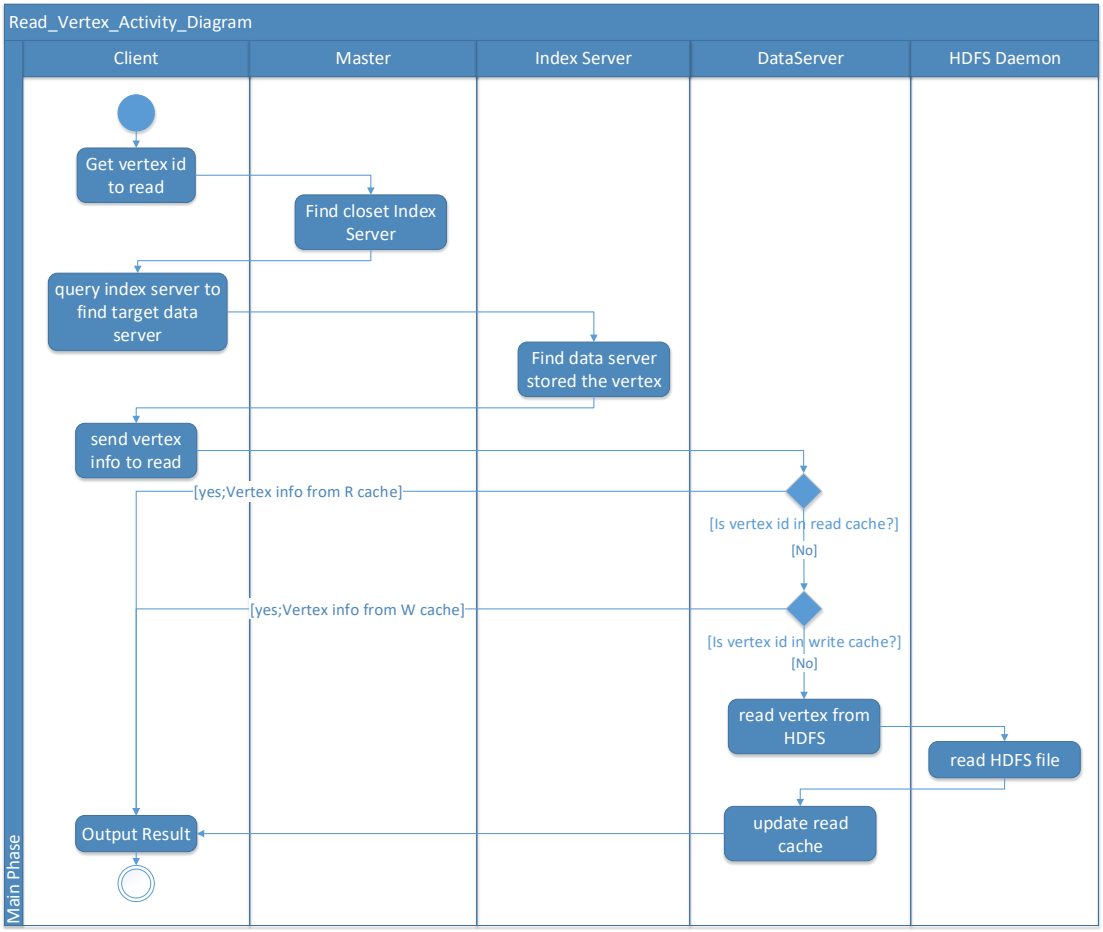
➢ The workflow of deleting vertex:



- The workflow of updating vertex:



- The workflow of querying vertex:


Read_Vertex_Activity_Diagram
Client
Master
Index Server
DataServer
HDFS Daemon
Main Phase
Get vertex id to read
Find closet Index Server
query index server to find target data server
Find data server stored the vertex
send vertex info to read
[yes;Vertex info from R cache]
[Is vertex id in read cache?]
[No]
[yes;Vertex info from W cache]
[Is vertex id in write cache?]
[No]
read vertex from HDFS
read HDFS file
update read cache
Output Result

7. Conclusion and further work

7.1 Conclusions

With the rapid development of IT technologies, most organizations have their own IT system to record different kinds of data in an organization. Individuals also collect and share data through the Internet. How to manage such a large amount of data and how to make a good use of it becomes a key challenge in knowledge management. As datasets become larger and more complex the ability to process them efficiently without loss or misinterpretation of information becomes a major challenge with current knowledge management systems. Especially in enterprise application, some real-time fraud detection in transactions in online payment company are suffer a lot from these.

The thesis describes a knowledge management system, which attempts to overcome these problems by developing system architecture and algorithms which will enable distributed data storage and parallel processing. The proposed system architecture consists of two parts: storage subsystem and knowledge clustering subsystem.

The proposed storage system focuses on two key issues: avoidance of data duplication, and optimization for parallel processing. Since the volume of the dataset may be very large, the storage sub-system has to avoid data duplication and needs to be located quickly. This is achieved by the use of multi dataset schemas to describe the dataset, and index the dataset. The index associated with each schema enables the data to be located rapidly. In order to optimize for parallel processing, distributed storage and index technologies are incorporated into the system.

In the knowledge clustering subsystem, a heterogeneous graph is used to describe a body of knowledge with the node representing individual components of the knowledge and the edge representing the linkage between those components. The thesis proposes a feature selection model, combines the graph attribute and graph structure together. The model for the heterogeneous

knowledge graph can deal with the incomplete attributes across knowledge and different types of link, according to a user specified attribute parameters.

The thesis gives an example of a prototype knowledge management system for fraud detection to combine all the ideas together and evaluate all the ideas proposed.

7.2 Further work

How to make a good use of massive data is still a big challenge in real world. The knowledge management system proposed is still in the first stage. A lot of further research work can be done in this area:

- The index to the distributed graph is still in the dataset level, we can make the index on the sematic level or knowledge level, which can support for sematic usage of dataset.
- The data structure for index is the hash table, distributed tree methods can be used here to gain better searching performance.
- The Mutual information based future selection method can be enhanced by involving multi-relation for more than two attributes
- The Non-Euclidean geometry weighting algorithm can be enhanced by changing the definition of angle δ

Reference

Airoldi, E. M., et al. (2008). "Mixed membership stochastic blockmodels." The Journal of Machine Learning Research **9**: 1981-2014.

Angelis, L. and I. Stamelos (2000). "A simulation tool for efficient analogy based cost estimation." Empirical Software Engineering **5**(1): 35-68.

Apache (2009). http://hbase.apache.org/.

Auer, M., et al. (2006). "Optimal project feature weights in analogy-based cost estimation: Improvement and limitations." Software Engineering, IEEE Transactions on **32**(2): 83-92.

Backstrom, L., et al. (2006). Group formation in large social networks: membership, growth, and evolution. Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM.

Bardsiri, V. K., et al. (2012). "A PSO-based model to increase the accuracy of software development effort estimation." Software Quality Journal: 1-26.

Basu, S., et al. (2004). A probabilistic framework for semi-supervised clustering. Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM.

Battiti, R. (1994). "Using mutual information for selecting features in supervised neural net learning." Neural Networks, IEEE Transactions on **5**(4): 537-550.

Bell, G., et al. (2006). "Petascale computational systems." Computer **39**(1): 110-112.

Blanas, S., et al. (2010). A comparison of join algorithms for log processing in mapreduce. Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM.

Bortner, D. and J. Han (2010). Progressive clustering of networks using structure-connected order of traversal. Data Engineering (ICDE), 2010 IEEE 26th International Conference on, IEEE.

Chakrabarti, D., et al. (2006). Evolutionary clustering. Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM.

Chang, F., et al. (2008). "Bigtable: A distributed storage system for structured data." ACM Transactions on Computer Systems (TOCS) **26**(2): 4.

Chen, R., et al. (2010). Large graph processing in the cloud. Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM.

Chesbrough, H. W. (2003). Open innovation: The new imperative for creating and profiting from technology, Harvard Business Press.

Chi, Y., et al. (2007). Evolutionary spectral clustering by incorporating temporal smoothness. Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM.

Christensen, J. F., et al. (2005). "The industrial dynamics of Open Innovation—Evidence from the transformation of consumer electronics." Research policy **34**(10): 1533-1549.

Clauset, A., et al. (2004). "Finding community structure in very large networks." Physical review E **70**(6): 066111.

Clerc, M. (2010). "Beyond standard particle swarm optimisation." International Journal of Swarm Intelligence Research (IJSIR) **1**(4): 46-61.

Condie, T., et al. (2010). MapReduce Online. NSDI.

Cooper, B. F., et al. (2008). "PNUTS: Yahoo!'s hosted data serving platform." Proceedings of the Vldb Endowment **1**(2): 1277-1288.

Cover, T. M. and J. A. Thomas (2012). Elements of information theory, John Wiley & Sons.

Das, S., et al. (2010). Ricardo: integrating R and Hadoop. Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM.

De Maesschalck, R., et al. (2000). "The mahalanobis distance." Chemometrics and Intelligent Laboratory Systems **50**(1): 1-18.

Dean, J. and S. Ghemawat (2008). "MapReduce: simplified data processing on large clusters." Communications of the ACM **51**(1): 107-113.

Dean, J. and S. Ghemawat (2010). "MapReduce: a flexible data processing tool." Communications of the ACM **53**(1): 72-77.

Estévez, P. A., et al. (2009). "Normalized mutual information feature selection." Neural Networks, IEEE Transactions on **20**(2): 189-201.

Fienberg, S. E. and M. E. Martin (1985). Sharing research data, Natl Academy Pr.

Group, I. S. B. S. (2003). "International Software Benchmarking Standards Group Data Disk, Release 8." 2013, from http://www.isbsg.org/.

Gruber, F. C. a. J. D. a. S. G. a. W. C. H. a. D. A. W. a. M. B. a. T. C. a. A. F. a. R. E. (2006). Bigtable: A distributed storage system for structured data. IN PROCEEDINGS OF THE 7TH CONFERENCE ON USENIX SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION - VOLUME 7.

Hive (2011). http://hive.apache.org/.

Huang, S.-J. and N.-H. Chiu (2006). "Optimization of analogy weights by genetic algorithm for software effort estimation." Information and software technology **48**(11): 1034-1045.

Jain, A. K. and R. C. Dubes (1988). Algorithms for clustering data, Prentice-Hall, Inc.

Kennedy, J. (2010). Particle swarm optimization. Encyclopedia of Machine Learning, Springer**:** 760-766.

Kim, M.-S. and J. Han (2009). "A particle-and-density based evolutionary clustering method for dynamic networks." Proceedings of the VLDB Endowment **2**(1): 622-633.

Kwak, N. and C.-H. Choi (2002). "Input feature selection for classification problems." Neural Networks, IEEE Transactions on **13**(1): 143-159.

Lakshman, A. and P. Malik (2009). Cassandra: structured storage system on a p2p network. Proceedings of the 28th ACM symposium on Principles of distributed computing, ACM.

Leskovec, J., et al. (2008). Statistical properties of community structure in large social and information networks. Proceedings of the 17th international conference on World Wide Web, ACM.

Li, Y.-F., et al. (2009). "A study of project selection and feature weighting for analogy based software cost estimation." Journal of Systems and Software **82**(2): 241-252.

Long, B., et al. (2007). A probabilistic framework for relational clustering. Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM.

Mahalanobis, P. C. (1936). "On the generalized distance in statistics." Proceedings of the National Institute of Sciences (Calcutta) **2**: 49-55.

Malewicz, G., et al. (2010). Pregel: a system for large-scale graph processing. Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM.

Mei, Q., et al. (2008). Topic modeling with network regularization. Proceedings of the 17th international conference on World Wide Web, ACM.

Mendes, E., et al. (2003). "A comparative study of cost estimation models for web hypermedia applications." Empirical Software Engineering **8**(2): 163-196.

Peng, H., et al. (2005). "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy." Pattern Analysis and Machine Intelligence, IEEE Transactions on **27**(8): 1226-1238.

Poli, R., et al. (2007). "Particle swarm optimization." Swarm intelligence **1**(1): 33-57.

Rigby, D. and C. Zook (2002). "Open-market innovation." Harvard business review **80**(10): 80-93.

Sayyad Shirabad, J. a. M., T.J. (2005). "The {PROMISE} Repository of Software Engineering Databases.". from http://promise.site.uottawa.ca/SERepository.

Shiga, M., et al. (2007). A spectral clustering approach to optimally combining numericalvectors with a modular network. Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM.

Stefansson, G. (2002). "Business-to-business data sharing: A source for integration of supply chains." International journal of production economics **75**(1): 135-146.

Stonebraker, M. (1986). "The case for shared nothing." IEEE Database Eng. Bull. **9**(1): 4-9.

Sun, Y., et al. (2009). itopicmodel: Information network-integrated topic modeling. Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on, IEEE.

Sun, Y., et al. (2009). Ranking-based clustering of heterogeneous information networks with star network schema. Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM.

Taskar, B., et al. (2001). Probabilistic classification and clustering in relational data. International Joint Conference on Artificial Intelligence, LAWRENCE ERLBAUM ASSOCIATES LTD.

Thang, N. D. and Y.-K. Lee (2010). An improved maximum relevance and minimum redundancy feature selection algorithm based on normalized mutual information. Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium on, IEEE.

Venugopal, S., et al. (2006). "A taxonomy of data grids for distributed data sharing, management, and processing." ACM Computing Surveys (CSUR) **38**(1): 3.

Von Luxburg, U. (2007). "A tutorial on spectral clustering." Statistics and computing **17**(4): 395-416.

Wang, C., et al. (2010). MapDupReducer: detecting near duplicates over massive datasets. Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM.

Yang, H.-c., et al. (2007). Map-reduce-merge: simplified relational data processing on large clusters. Proceedings of the 2007 ACM SIGMOD international conference on Management of data, ACM.

Yang, T., et al. (2009). Combining link and content for community detection: a discriminative approach. Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM.

Zhou, Y., et al. (2009). "Graph clustering based on structural/attribute similarities." Proceedings of the VLDB Endowment **2**(1): 718-729.